Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

Clean architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Albert Juan MONTES ANCCASI

ASESOR

Cayo Víctor LEÓN FERNÁNDEZ

Lima, Perú

2018

Referencia bibliográfica

Montes, A. (2018). *Clean architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD*. [Trabajo de suficiencia profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.

#214

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
Escuela Profesional de Ingeniería de Sistemas

Acta de Sustentación del Trabajo de Suficiencia Profesional

Siendo las 18:00 horas del día 17 de diciembre del año 2018, se reunieron los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Ing. Fermín Pérez Félix Armando (Presidente), Dra. Pró Concepción Luzmila Elisa (Miembro) y el Mg. León Fernandez Cayo Víctor (Miembro Asesor) para la sustentación del Trabajo de Suficiencia Profesional Intitulado: ***"CLEAN ARCHITECTURE PARA MEJORAR EL DESARROLLO DE APLICACIONES MOVILES EN LA EMPRESA GMD",*** *por el Bachiller:* ***Montes Anccasi, Albet Juan****; para obtener el Título Profesional de Ingeniero de Software.*

Acto seguido de la exposición del Trabajo de Suficiencia Profesional, el Presidente invitó al Bachiller a dar las respuestas a las preguntas establecida por los miembros del Jurado.

El Bachiller en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez a las observaciones y preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el Bachiller obtuvo la nota de ...16... (En letras) ... DIECISEIS

A continuación el presidente del jurado el Ing. Fermín Pérez Félix Armando, declara al Bachiller Ingeniero de Software

Siendo las 18:30 horas, se levantó la sesión.

Presidente
Ing. Fermín Pérez Félix Armando

Miembro
Dra. Pró Concepción Luzmila Elisa

Miembro Asesor
Mg. León Fernandez Cayo Víctor

FICHA CATALOGRÁFICA

CLEAN ARCHITECTURE PARA MEJORAR EL DESARROLLO DE APLICACIONES MOVILES EN LA EMPRESA GMD

AUTOR: MONTES ANCCASI, ALBERT JUAN

ASESOR: LEÓN FERNANDEZ, CAYO VICTOR

LIMA – PERÚ, 2018

Título Profesional/Grado Académico: Título Profesional de Ingeniero de Software
Área/Programa/Línea de Investigación: Ingenierías / Tecnología de Información y Comunicación / Ingeniería de Software

Pregrado: Universidad Nacional Mayor de San Marcos – Facultad de Ingeniería de Sistemas e Informática – Escuela Profesional de Ingeniería de Software

Formato 28 x 20 cm Páginas: xii,99

DEDICATORIA

A mis padres, por su amor, trabajo y apoyo incondicional en cada etapa de mi vida, gracias a ustedes hemos logrado llegar hasta aquí.

AGRADECIMIENTOS

Gracias a mi asesor Cayo León Fernandez, por su dedicación y motivación. Ha sido un privilegio poder contar con su guía.

Gracias a la Comisión del Programa de Titulación por Trabajo de Suficiencia Profesional de la Escuela Profesional de Ingeniería de Sistemas y de la Escuela Profesional de Ingeniería de Software de la Facultad de Ingeniería de Sistemas e Informática de la UNMSM, por facilitar los mecanismos administrativos y educativos para posibilitar el acceso a los bachilleres en Ingeniería de Sistemas y en Ingeniería de Software para la elaboración del Informe de trabajo de suficiencia profesional; y así preparar a los bachilleres en la obtención del Título Profesional de Ingeniero de Sistemas y de Ingeniero de Software.

Gracias a los compañeros bachilleres en Ingeniería de Sistemas y en Ingeniería de Software, por su compañerismo, dedicación, apoyo, colaboración y conocimiento; lo que nos permitió afrontar con éxito los trabajos grupales, presentaciones, exámenes y reuniones que posibilitaron aprobar satisfactoriamente los cursos que formaron parte del Programa de Titulación por Trabajo de Suficiencia Profesional.

Finalmente, gracias a mi familia y amigos, sin quienes este esfuerzo no hubiera sido posible.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

CLEAN ARCHITECTURE PARA MEJORAR EL DESARROLLO DE APLICACIONES MOVILES EN LA EMPRESA GMD

Autor: **Montes Anccasi, Albert Juan**

Asesor: **León Fernández, Cayo Víctor**

Título: **Informe de Trabajo de Suficiencia Profesional para optar el Título Profesional de Ingeniero de Software**

Fecha: **Diciembre 2018**

RESUMEN

El presente informe de experiencia profesional describe la mejora en el desarrollo de aplicaciones móviles usando Clean Architecture en la empresa GMD. En el año 2017 se detectó que, en los procesos de mantenimiento, mejora y prueba de los aplicativos móviles se estaba demandando mayor uso de recursos, debido a que los aplicativos con el tiempo crecen y su complejidad también se incrementaba. En este sentido fue necesario implementar una arquitectura que permita mayor escalabilidad, mantenimiento, mejora continua y pruebas de los aplicativos móviles. Para cumplir con este propósito, el área de innovación conformó un equipo de analista programadores para implementar Clean Architecture en los proyectos móviles Android. Las etapas desarrolladas fueron: Investigar y analizar arquitecturas para móviles (Android), Elegir la arquitectura, Codificar un aplicativo Android con la arquitectura y por último Validar, comunicar y documentar los resultados.

Obteniéndose los resultados esperados por GMD, se evidencia de esta manera que los proyectos en los cuales se implementó Clean Architecture permitieron obtener mayor escalabilidad, mantenimiento, mejora continua y pruebas con mayor eficiencia.

Palabras claves: Arquitectura de software, clean architecture, SOLID, mejoras en el desarrollo de software, aplicaciones móviles.

MAJOR NATIONAL UNIVERSITY OF SAN MARCOS

FACULTY OF SYSTEMS AND COMPUTER ENGINEERING

PROFESSIONAL SCHOOL OF SOFTWARE ENGINEERING

CLEAN ARCHITECTURE TO IMPROVE THE DEVELOPMENT OF MOBILE APPLICATIONS IN THE GMD COMPANY

Author: **Montes Anccasi, Albert Juan**

Advisor: **León Fernández, Cayo Víctor**

Title: **Professional Sufficiency Work Report to choose the Professional Title of Software Engineer**

Date: **December 2018**

ABSTRACT

This professional experience report describes the improvement in the development of mobile applications using Clean Architecture in the company GMD. In the year 2017 it was detected that, in the processes of maintenance, improvement and testing of mobile applications, greater use of resources was being demanded, due to the fact that the applications grow over time and their complexity also increased. In this sense, it was necessary to implement an architecture that allows greater scalability, maintenance, continuous improvement and testing of mobile applications. To fulfill this purpose, the innovation area formed a team of analyst programmers to implement Clean Architecture in Android mobile projects. The stages developed were: Investigate and analyze architectures for mobile (Android), Choose the architecture, Codify an Android application with the architecture and finally Validate, communicate and document the results.

Obtaining the results expected by GMD, it is evident in this way that the projects in which Clean Architecture was implemented allowed obtaining greater scalability, maintenance, continuous improvement and tests with greater efficiency.

Keywords: Software architecture, clean architecture, SOLID, improvements in software development, mobile applications.

INDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El presente informe de experiencia profesional describe la implementación de Clean Architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD. El 2017 se detectó en los procesos de mantenimiento, mejora y prueba de los aplicativos móviles se estaba demandando mayor uso de recursos, esto debido a que los aplicativos con el tiempo implementan nuevas funcionalidades y por ende la complejidad también se incrementaba. Entonces fue necesario plantear objetivos para mitigar los riesgos mencionados.

Previo a la implementación de Clean Architecture, la empresa siempre busco incorporar buenas prácticas y recomendaciones en el desarrollo; inicialmente se implementó una arquitectura tradicional MVC (Modelo Vista Controlador), posteriormente se implementó el patrón de diseño MVP (Modelo Vista Presentación). En este sentido la propuesta de implementar Clean Architectura se alinea a los objetivos de mejora continua de GMD. Clean Architecture es una arquitectura propuesta por Robert Martin en un blog de tecnología en el 2012 y posteriormente en el 2018 publicó un libro más detallado; está arquitectura permite atribuir al software: independencia de los Frameworks, facilitar las pruebas, independencia de la interfaz gráfica (UI), Independencia de la base de datos e independencia de cualquier agente externo.

Los proyectos en los cuales se implementó Clean Architecture, cumplieron con los objetivos de GMD, se evidencia una mejor distribución de software (código, clases, paquetes y módulos) lo que permitió entender mejor los algoritmos, flujo de datos, dependencias y responsabilidades de cada parte del software; se redujo y pudo predecir mejor el costo del mantenimiento de los proyectos; las mejoras en los proyectos fueron más sencillas y enfocadas a los casos de uso, por último, las pruebas unitarias, modulares e integrales fueron más fáciles de realizar.

En conclusión, los proyectos que implementaron Clean Architecture, en el transcurso del ciclo de vida de software pueden incrementar funcionalidad, cambiar de origen de datos, rediseñar las interfaces, actualizar las librerías externas y realizar pruebas en distintos entornos (desarrollo, producción); sin tener mayores dificultades y reduciendo los costos a largo plazo.

El presente informe está estructurado en cinco capítulos de la siguiente manera: En el Capítulo I, contiene mi trayectoria profesional, la que refleja la experiencia adquirida y que permitió que pueda realizar de manera adecuada el proyecto "Clean Architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD". El Capítulo II, contiene el contexto en el que se desarrolló la experiencia, descripción de la empresa GMD; área, cargo y funciones desempeñadas, así como la experiencia profesional realizada en la organización. En el Capítulo III, describo el problema presentado, los objetivos y alcance, las etapas y metodología empleadas, los fundamentos utilizados e implementación, así como la evaluación económica de la solución. En el Capítulo IV, se expone la reflexión crítica de la experiencia y finalmente el Capítulo V, contiene las conclusiones y recomendaciones.

CAPITULO I - TRAYECTORIA PROFESIONAL

PRESENTACION PROFESIONAL

Soy BACHILLER en INGENIERIA DE SOFTWARE de la Universidad Nacional Mayor de San Marcos, Tengo más de 2 años de experiencia profesional y 2 de pre profesional en organismos públicos y empresas privadas. En la actualidad me desempeño como Analista Programador de aplicaciones móviles en GMD (consultora de TI), desde agosto de 2017. Tengo amplio conocimiento en desarrollo de soluciones móviles.

Mi experiencia profesional se da con especial énfasis en los siguientes aspectos:

- Análisis, diseño, desarrollo y mantenimiento aplicaciones móviles.
- Evaluación, diagnóstico y propuesta de arquitecturas para aplicaciones móviles.
- Desarrollo de servicios web y su integración con otros sistemas.
- Análisis y construcción de soluciones web.
- Trabajo sobre metodología de desarrollo Scrum.

Busco contribuir con mi experiencia y conocimiento en los equipos que integro, siendo eficiente, eficaz y permanentemente proactivo en mi desempeño profesional.

FORMACION ACADEMICA

EDUCACION SUPERIOR: GRADOS ACADÉMICOS	
Bachiller en Ingeniería de Software Universidad Nacional Mayor de San Marcos – Facultad de Ingeniería de Sistemas e Informática – Escuela Académico Profesional de Ingeniería de Software.	2016
Egresado en Ingeniería de Software Universidad Nacional Mayor de San Marcos – Facultad de Ingeniería de Sistemas e Informática – Escuela Académico Profesional de Ingeniería de Software.	2009 – 2014

Cursos	
Nombre: ANDROID AVANZADO Desarrollo para aplicaciones móviles. **Institución:** Academias Móviles	Agosto - Setiembre 2018
Nombre: Aplicaciones Móviles – Android **Institución:** Área 51	Julio 2016
Nombre: Modulo I: Interpretación de los requisitos de la norma NTP-ISO/IEC 27001:2014 **Institución:** Argos Consulting Group	Marzo 2016
Nombre: Modulo II: Herramientas para la gestión de riesgos basado en la norma ISO 31000:2009 **Institución:** Argos Consulting Group	Marzo 2016
Nombre: Modulo III: Formación de Auditores Internos para la auditoría del sistema de Gestión de la Seguridad de la Información NTP ISO/IEC 27001:2014 **Institución:** Argos Consulting Group	Abril 2016
Nombre: Capacitación y Formación de Auditores Internos en el sistema de Gestión de la seguridad de la Información NTP ISO (IEC 27001:2014) **Institución:** Argos Consulting Group	Marzo-abril 2016
Nombre: Capacitación en tecnología PKI – Firma Digital **Institución:** Soft&Net soluciones en tecnología	Junio 2015
Nombre: Administración de Hardware Security Module (HSM) y Uso de Api de Programación **Institución:** Soft&Net soluciones en tecnología	Octubre 2016
Nombre: Manejo de la consola de administración y uso básico y avanzado de los componentes y funcionalidades de la solución colaborativa Microsoft office 365 **Institución:** Red Redes y Servicios	Noviembre 2016

Nombre: Curso de programación en Android **Institución:** Platzi	Junio 2015
Nombre: Herramientas Informáticas **Institución:** Universidad Nacional Mayor de San Marcos	Mayo – junio 2010

EXPERIENCIA PROFECIONAL

De enero 2013 hasta noviembre 2014	**Cargo:** Desarrollador Web y Móvil. **Nombre de la empresa:** KILLARI BYTE SAC. **Descripción de trabajo desarrollado:** • Análisis, diseño y desarrollo de aplicaciones móviles en plataformas Android. • Análisis, diseño y desarrollo de aplicaciones Web. • Análisis, diseño y desarrollo de Servicios Web. • Análisis, diseño y desarrollo de Video juegos con JavaScript.
De octubre 2014 hasta mayo 2015	**Cargo:** Servicio de desarrollo de aplicaciones para equipos móviles. **Nombre de la empresa:** Programa nacional de becas y créditos educativos - PRONABEC. **Descripción de trabajo desarrollado:** • Análisis, diseño y desarrollo de aplicaciones móviles en plataformas Android. • Análisis, diseño y desarrollo de servicios web.
De agosto 2015 hasta marzo 2017	**Cargo:** Analista programador para equipos móviles. **Nombre de la empresa:** Programa nacional de becas y créditos educativos - PRONABEC. **Descripción de trabajo desarrollado:** • Análisis, diseño y desarrollo de aplicaciones móviles en plataformas Android. • Desarrollo y soporte de aplicativos administrativos web en .net.

	• Soporte de la plataforma de firma digital con certificados digitales. **Logros:** • Desarrollo del aplicativo móvil "Intranet del becario", para mejorar el acceso a la información de los becarios. • Desarrollo del aplicativo móvil "Intranet del trabajador", para mejorar el acceso a la información de los trabajadores. • Desarrollo de la plataforma de seguimiento al becario a través de encuestas y cuestionarios personalizados. • Participación como auditor interno para la auditoría del sistema de gestión de seguridad de la información NTP ISO/IEC 27001:2014 • Participación con el proyecto Intranet del becario en el Concurso de Buenas practica en la gestión pública 2016. • Desarrollo de servicios web para la integración con los aplicativos móviles (Intranet del becario y trabajador).
De abril 2017 hasta agosto 2017	**Cargo:** Analista Programador. **Nombre de la empresa:** KILLARI BYTE. **Descripción de trabajo desarrollado:** • Análisis, diseño y desarrollo de aplicaciones móviles en plataformas Android. • Análisis, diseño y desarrollo de servicios web.
De agosto 2017 hasta la fecha	**Cargo:** Analista Programador. **Nombre de la empresa:** Inversiones palo Alto SAC. (**GMD**). **Descripción de trabajo desarrollado:** • Análisis, diseño y desarrollo de aplicaciones móviles en plataformas Android. • Análisis, diseño y desarrollo de servicios web. • Implementación de la arquitectura de desarrollo para aplicativos Android.

	Logros: • Clean Architecture para el desarrollo de aplicaciones móviles Android. • Desarrollo de Pagalo.pe, aplicativo del banco de la Nación para el pago de tasas, pago de servicios y transferencias. (módulo de servicios y transferencias) • Desarrollo de Gente BN, aplicativo móvil de comunicación interna de banco de la nación. • Desarrollo de Tancadas, aplicativo móvil de para suministrar agua a sistemas para SEDAPAL. • Desarrollo de FreeBuy, aplicativo de compras a través de lectura de código de barras y pagos con tarjeta de crédito. • Mantenimiento y desarrollo de nuevas tasas y entidades en el aplicativo móvil "Págalo" de banco de la nación.

CAPITULO II - CONTEXTO EN EL QUE SE DESAROLLO LA EXPERIENCIA

2.1 Empresa - Actividad que realizará.

GMD es una empresa de Outsourcing de Procesos de Negocios, Tecnología de la Información (TI) y Transformación Digital, ubicada en Lima – Perú, con más de 33 años de experiencia y un staff de 3000 profesionales.

En 1985, la empresa GMD inicia sus operaciones y a lo largo de los años ha crecido gracias a su estrategia de Servicios de Outsourcing de TI y de Procesos de Negocio lo que ha permitido convertirse en la empresa líder de Outsourcing en el Perú.

La prestigiosa firma "LLoyd´s Register Quality Assurance", reconoce la calidad de los servicios que brinda la empresa GMD a sus clientes y dispone otorgarle la certificación Internacional de calidad ISO 9001; del mismo modo y sumándose los reconocimientos, en el año 2015, el "European Software Institute (ESI)", le otorga la certificación CMMI-5 (Capability Maturity Model Integration), estándar internacional).

El 06 de junio de 2017 la firma norteamericana Advent International Corporation un fondo de Inversión de capital Privado que cuenta con un portafolio de activos de más de 42 mil millones de dólares, el 9 de septiembre de 2017 adquirió una participación mayoritaria, pasando a ser GMD ahora una empresa del portafolio de Advent International, el 6 de septiembre de 2018 GMD cambia de imagen y da surgimiento a ***"CANVIA".***

2.2 Visión

Ser la empresa referente en servicios de Transformación Digital en la Región Pacífico.

2.3 Misión

Proveer soluciones de Transformación Digital que aporten valor al negocio de nuestros clientes en base al conocimiento de industria, reconocidos por nuestra calidad de servicio, excelencia operacional y el talento de nuestros colaboradores.

2.4 Valores

- ✓ Orientación de Servicio
- ✓ Colaboración
- ✓ Innovación
- ✓ Excelencia

2.5 Organigrama de la Empresa

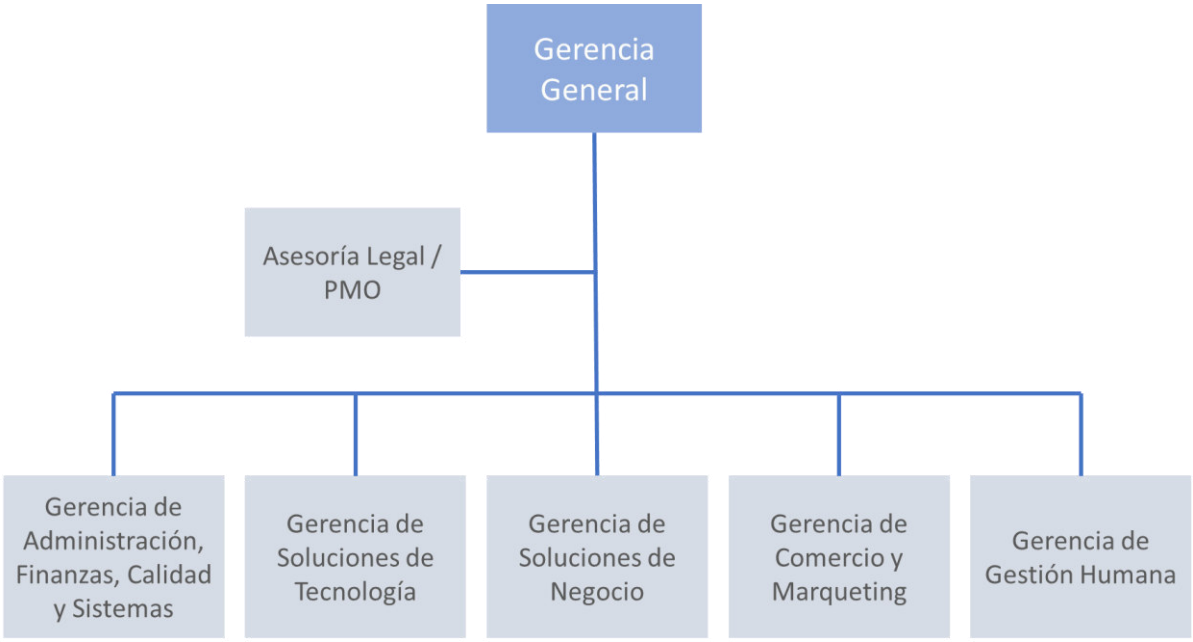


Figura 1 Organigrama de GMD

Fuente: (GMD, 2017). Obtenido de www.gmd.com.pe

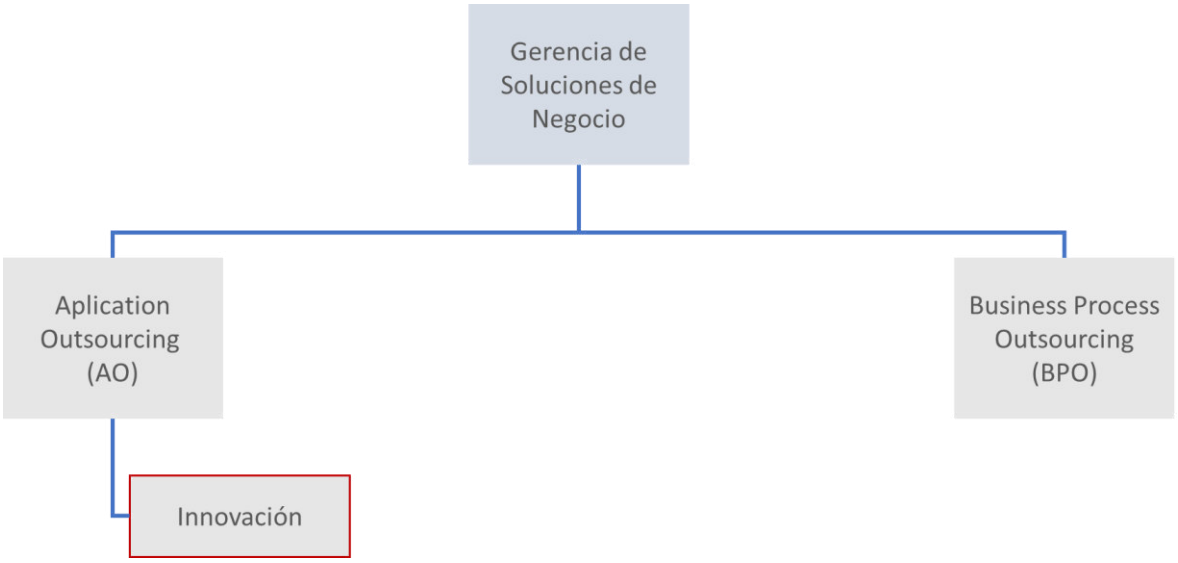


Figura 2 Organigrama de Gerencia de Soluciones de Negocio (GMD)

Fuente: (GMD, 2017). Obtenido de www.gmd.com.pe

Orgánicamente, la empresa GMD está compuesta por cinco Gerencias, un área que integra asesoría legal y la oficina de gestión de proyectos (PMO), ver *Figura 1.*

Las cinco Gerencias cuentan con una estructura orgánica y denominación diferente; una de ellas, es la Gerencia de Soluciones de Negocio, ver *Figura 2,* que se encarga de gestionar los proyectos y/o requerimientos de tecnología de información; asimismo, es responsable de asegurar el soporte transaccional a las operaciones y estrategias del negocio de los clientes; a su vez, esta Gerencia cuenta con dos áreas:

Application Outsourcing (AO): Se encarga del desarrollo de aplicaciones y servicios de consultoría y está compuesta por diversos proyectos que están conformados por diferentes especialistas encargados de atender los requerimientos del mismo. Uno de estos proyectos es el denominado “Innovación”, que se especializa en el desarrollo a aplicaciones móviles y de nuevas tecnologías. Cabe mencionar que mi persona, ha desarrollado su experiencia profesional en este proyecto, pues he implementado Clean Arquitectura en los proyectos Android, lo cual es materia de análisis y redacción del presente informe.

Business Process Outsourcing (BPO): Realiza la subcontratación de funciones de procesos de negocios que soliciten los clientes.

2.6 Área, Cargo y Funciones desempeñadas

Desde el 09 de agosto de 2017 hasta la fecha (septiembre 2018), me desempeño como Analista Programador en el proyecto "Innovación" (se especializa en desarrollo soluciones móviles y nuevas tecnologías), adscrito al área de Application Outsourcing (OA) de la Gerencia de Soluciones de Negocio de la empresa GMD, y las funciones que desempeño son las siguientes:

a) Desarrollo y mantenimiento de aplicativos móviles en Android.
b) Estudio, evaluación y propuesta de arquitecturas móviles.
c) Desarrollo, actualización y mantenimiento de la arquitectura de proyectos desarrollados en GMD.
d) Definición y desarrollo de servicios web.
e) Elaboración de diagnóstico de aplicaciones móviles.

CAPITULO III - ACTIVIDADES DESARROLLADAS

3.1 Situación Problemática

3.1.1 Definición Del Problema

Desde el inicio de actividades la empresa GMD en el mercado, ha venido desarrollando estrategias efectivas para satisfacer las necesidades de sus clientes, en el año 2017 se detectó que en las fases mantenimiento, mejora y prueba de los aplicativos móviles se estaba demandando mayor uso de recursos (tiempo y personal), debido a que en el transcurso del ciclo de vida del software se incrementó funcionalidades, cambio de origen de datos, rediseñó las interfaces, actualizó librerías externas y/o realizó pruebas en distintos entornos (desarrollo, producción).

Las consecuencias de lo mencionado son varias:

- Impacto negativo en la calidad de software.
- Mayor tiempo en la implementación de nuevas funcionalidades.
- Mayor tiempo en realizar cambios en el software.
- Poca facilidad para testear.
- Muchos códigos duplicados.
- Mayor tiempo en entender el código de otro desarrollador.

En este sentido el problema que evaluamos fue que la arquitectura tradicional (MVC) no cumplía con las necesidades de mantener un software, que permita mantener la calidad en todo el ciclo de vida del software, además de incrementar gradualmente los costos de mantenimiento y mejoras en cada nuevo ciclo.

3.2 Solución

3.2.1 Objetivo General

Implementar Clean Architecture para mejorar el desarrollo móvil en la empresa GMD.

3.2.2 Objetivos Específicos

- Evaluación de arquitectura en Android y mejoras.
- Elegir la Arquitectura para Android.
- Codificar un aplicativo Android Con Clean Architecture.
- Validar el aplicativo
- Comunicar y documentar.

3.2.3 Alcance

Definir una arquitectura para el desarrollo de aplicaciones móviles y desarrollar un aplicativo base en Android para empresas y organizaciones.

3.2.4 Etapas y Metodología

Para la implementación de Clean Architecture en GMD se desarrollaron múltiples reuniones de coordinación con los interesados (programadores y analistas programadores). A continuación, se describen las etapas.

- ✓ Evaluar arquitecturas en Android y validar que cumpla las necesidades.
- ✓ Elegir la arquitectura para Android.
- ✓ Codificar un aplicativo Android con la arquitectura elegida.
- ✓ Validar el aplicativo.
- ✓ Documentar y Comunicar.

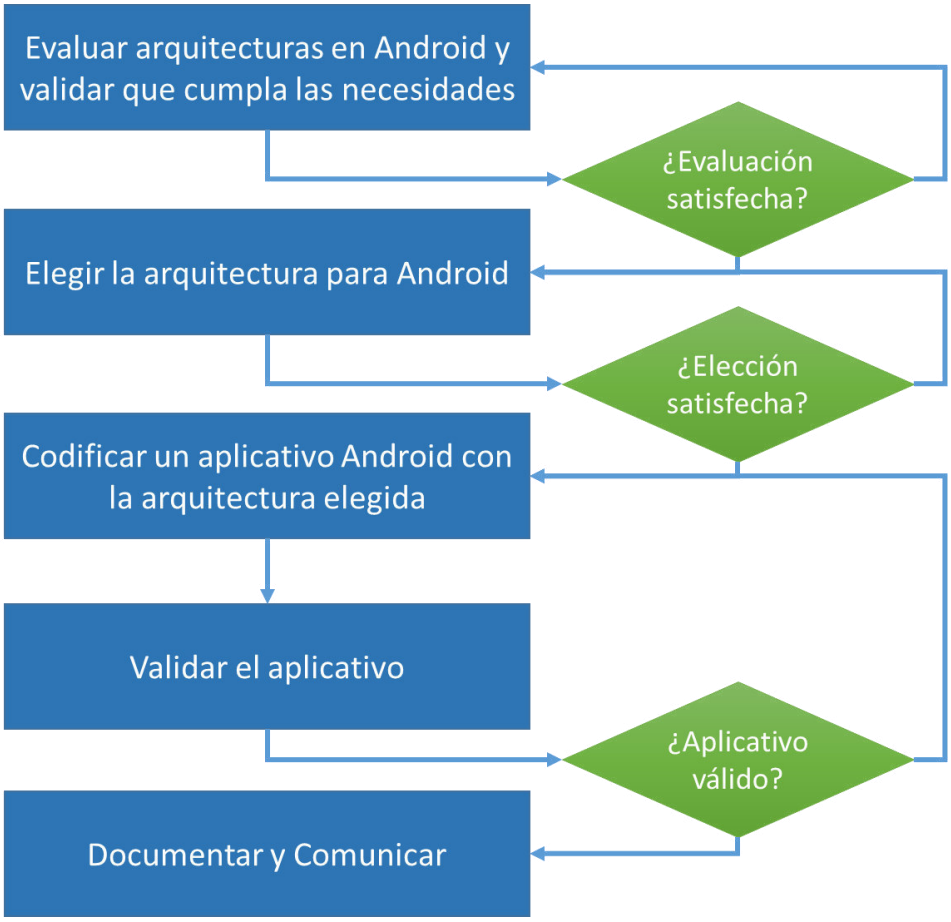


Figura 3. Etapas de la metodología. GMD

Fuente: Elaboración propia.

a) Evaluar arquitecturas en Android y validar que cumpla las necesidades:
Se realiza la evaluación de las distintas arquitecturas existentes para Android y a su vez comparando las ventajas y desventajas de cada una de ellas.

b) Elegir la arquitectura para Android:
En esta fase se elige la arquitectura que cumpla en mayor medida con las necesidades del problema.

c) Codificar un aplicativo Android con la arquitectura elegida:
En esta fase se desarrolla la plantilla en Android, para este fin se utilizó el entorno de desarrollo integrado (IDE) Android Studio.

d) Validar el aplicativo:
En esta etapa se valida la solución desarrollada, si es necesario se puede volver a la codificación para realizar ajustes.

e) Documentar y Comunicar:
En esta última etapa se documenta (manual técnico, lineamiento y empaquetamiento del código) y, por último, se comunicar al equipo los resultados obtenidos y muestra el desarrollo del proyecto base con Clean Architecture.

3.2.5 Fundamentos Utilizados y Marco Teórico.

3.2.5.1 Arquitectura:

James Marston Fitch: La arquitectura es un instrumento cuya función principal es la de intervenir en favor del hombre.

Glenn Murcutt: Necesitamos soluciones para los problemas reales, no inventar problemas para poder empatar con nuevas soluciones.

3.2.5.2 Arquitectura de Software:

La Arquitectura del Software (AS) es la parte de la ingeniería del software que se ocupa de la descripción y el tratamiento de un sistema como un conjunto de componentes, que facilite su organización en los diferentes subsistemas que lo forman. [Cle96 y IEEE 1471-2000] (Martin, Clean Architecture – A Craftsman´s Guide to Software Structure and Design, 2018)

Cuando hablamos de arquitectura de software muchos de nosotros tenemos un concepto muy parecido, pero no necesariamente satisface a todos en este campo. Quizás para tener una visión de conjunto, lo más factible sea revisar otras definiciones.

Más allá de los algoritmos y estructuras de datos de la computación, el diseño y especificación de la estructura global del sistema emerge como un nuevo tipo de problema. Los detalles estructurales incluyen: la organización general y la estructura de control global; los protocolos de comunicación, sincronización y acceso a datos; la asignación de funcionalidad a los elementos de diseño; la

distribución física; la composición de los elementos de diseño; su escalabilidad y los aspectos de rendimiento; y la selección entre alternativas de diseño.
(1993 por David Garlan y Mary Shaw [GaSh93]):

La arquitectura del software está compuesta por la estructura de los componentes de un programa o sistema, sus interrelaciones, y los principios y reglas que gobiernan su diseño y evolución a lo largo del tiempo
(David Garlan y Dewayne Perry [GaPe95])

La arquitectura del software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema, y las formas en que los componentes interactúan y se coordinan para alcanzar el objetivo del sistema. La vista arquitectónica es una vista abstracta de un sistema, aportando el más alto nivel de compresión y la supresión del detalle inherente a la mayor parte de abstracciones.
(Clements [Cle96])

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución. [IEEE 1471-2000]

La arquitectura del software captura y preserva las intenciones de los diseñadores sobre la estructura y el comportamiento de los sistemas, proporcionando un mecanismo de defensa contra el deterioro de los sistemas antiguos. Esta es la clave para lograr el control intelectual sobre la creciente complejidad de los sistemas….
… Paradójicamente… aún no hay un consenso sobre cuál es la respuesta a la pregunta de qué es la arquitectura del software y que sea ampliamente aceptada… Llegar a un acuerdo sobre la definición más adecuada para esta disciplina fue uno de los objetivos de definir el estándar IEEE. Sin embargo, esta falta de acuerdo no ha sido impedimento para que la arquitectura del software progrese. [2006 de IEEE]

¿Y de qué está hecho el software? A diferencia de los edificios, que pueden estar hechos de ladrillos, hormigón, madera, acero y vidrio, el software está hecho de software. Las grandes construcciones de software están hechas de componentes de software más pequeños, que a su vez están compuestos por componentes de software más pequeños, y así sucesivamente. (Martin, Clean Architecture – A Craftsman´s Guide to Software Structure and Design, 2018)

"La arquitectura de está ubicada en una fase previa al diseño, debido a que sus tareas son el fundamento del diseño de un sistema". (Booch 2017)

3.2.5.3 Clean Code

El código limpio es aquel código que está estructurado de forma compresible, que es claro en sus intenciones, fácil de leer, que es fácilmente mantenible y que está testeado. En el libro Clean Code de Martin van dando algunas ideas para conseguir escribir código limpio, hablando de principios SOLID. (Martin, Clean Code – A Handbook of Agile Software Craftsmanship, 2009)

3.2.5.4 SOLID:

SOLID es un acrónimo de los primeros cinco principios de diseño orientado a objetos (OOD) de Robert C. Martin.

Estos principios, cuando se combinan entre sí, facilitan que un programador desarrolle software que sea fácil de escalar, mantener y probar. También facilitan a los desarrolladores errores en el código, refactorizar fácilmente el código y también son parte del desarrollo de software ágil o adaptativo.

Que significa SOLID:

S - Single-responsiblity principle
O - Open-closed principle
L - Liskov substitution principle
I - Interface segregation principle
D - Dependency Inversion Principle

Singles responsiblity principle (Principio de responsabilidad única):

Significa que una clase debe tener solo una sola razón para existir, es decir no debemos de acumular todo el código en una sola clase, realizar consultas a la base desde la vista, etc.

Open-closed principle (Principio abierto cerrado)

Nos Indica que el código que se escribe debe estar abierto para la extensión, pero cerrado para la modificación, es decir podemos expandir la funcionalidad de un módulo sin tener que editar el código de este.

Liskov substitution principle (Principio de sustitución de Liskov)

Las clases que heredan deben seguir comportándose como su padre, es decir solo debes heredar de una clase para añadir comportamiento más para modificar el existente.

Interface segregation principle (Principio de segregación de interfaces)

Muchas interfaces cliente específicas son mejores que una interfaz de propósito general, indica que no debemos de dar más información de la necesaria para que le modulo o clase funcione,

Dependency Inversion Principle (Principio de Inversión de dependencias)

Debemos reducir la dependencia que existe entre módulos de nuestra aplicación, es decir los módulos no debes de ser los encargados de crear los objetos con los que trabajan, sino que alguien más debe crearlos y dárselos cuando lo necesiten.

(Martin, Clean Architecture – A Craftsman´s Guide to Software Structure and Design, 2018)

3.2.5.5 Ciclo de vida del desarrollo de software:

El ciclo de vida del desarrollo Software, es una secuencia estructurada y bien definida de las etapas en Ingeniería de software para desarrollar el producto software.

El software obtenido tras el proceso puede ser visto como el "producto" que entra al proceso, se transforma (a lo largo de la cadena de tareas) y que sale del proceso hasta obtener el producto deseado. Desde esta perspectiva del producto, se pueden establecer los estados por los que va pasando el producto en un proceso software: la entrada al proceso es una necesidad, que una vez estudiada se convierte en una especificación de requisitos, que posteriormente se transforma en un diseño del sistema, para pasar más adelante a ser un código y finalmente un sistema software completo e integrado. Este enfoque orientado al producto, focalizado en el producto transformado (en lugar del proceso que lo transforma) se llama ciclo de vida. Es decir, el ciclo que el producto software sufre a lo largo de su vida, desde que nace (o se detecta la necesidad) hasta que muere (o se retira el sistema). [Juristo Juzgado, N. b 1996].

La norma IEEE 1074 (Estándar IEEE del Ciclo de Vida para el Proceso de Desarrollo de Software) [IEEE, 1991] define ciclo de vida como: "una aproximación lógica de la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software".

La norma ISO 12207-1 (Proceso del Ciclo de Vida del Software) [ISO, 1994], define ciclo de vida como: "un marco de referencia, que contiene los procesos, las actividades y tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso".

3.2.5.6 Android:

Android es un sistema operativo para dispositivos móviles. Está basado en GNU/Linux e inicialmente fue desarrollado por Google. La presentación de la plataforma Android se realizó el 5 de noviembre de 2007 junto con la fundación Open Handset Alliance, un consorcio de 48 compañías de hardware, software y telecomunicaciones comprometidas a la promoción de estándares abiertos para dispositivos móviles.

Esta plataforma permite el desarrollo de aplicaciones por terceros (personas ajenas a Google), para lo cual, los desarrolladores deben de escribir código gestionado en el lenguaje de programación Java y controlar los dispositivos por medio de bibliotecas desarrolladas o adaptadas por Google, es decir, escribir programas en C u otros lenguajes, utilizando o no las bibliotecas de Google (compilándolas a código nativo de ARM). Sin embargo, este esquema de desarrollo no es oficialmente soportado por Google.

La mayoría del código fuente de Android ha sido publicado bajo la licencia de software Apache, una licencia de software libre y código fuente abierto.

Android es una pila de software de código abierto basado en Linux creada para una variedad amplia de dispositivos y factores de forma. En el siguiente diagrama se muestran los componentes principales de la plataforma Android.

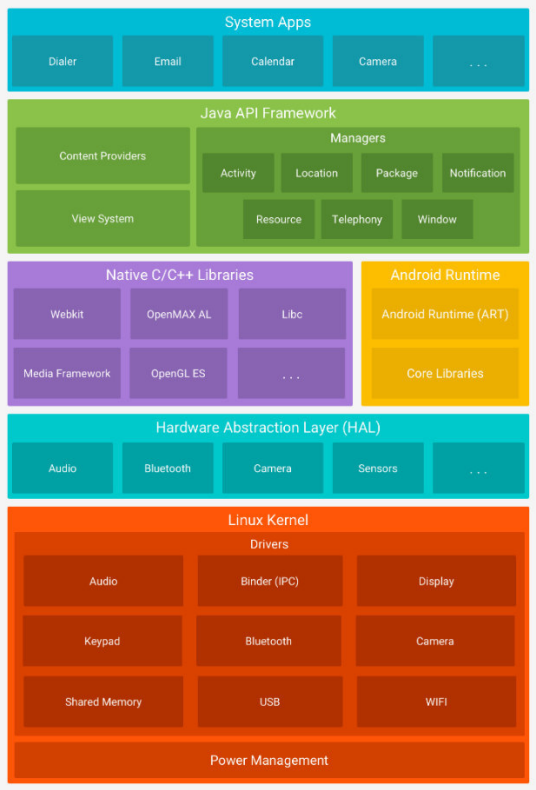


Figura 4 Arquitectura de la plataforma Android

Fuente: Android. (22 de 08 de 2018). Android Developer. Obtenido de developer.android.com: https://developer.android.com/guide/platform/?hl=es-419

3.2.5.7 Clean Architecture:

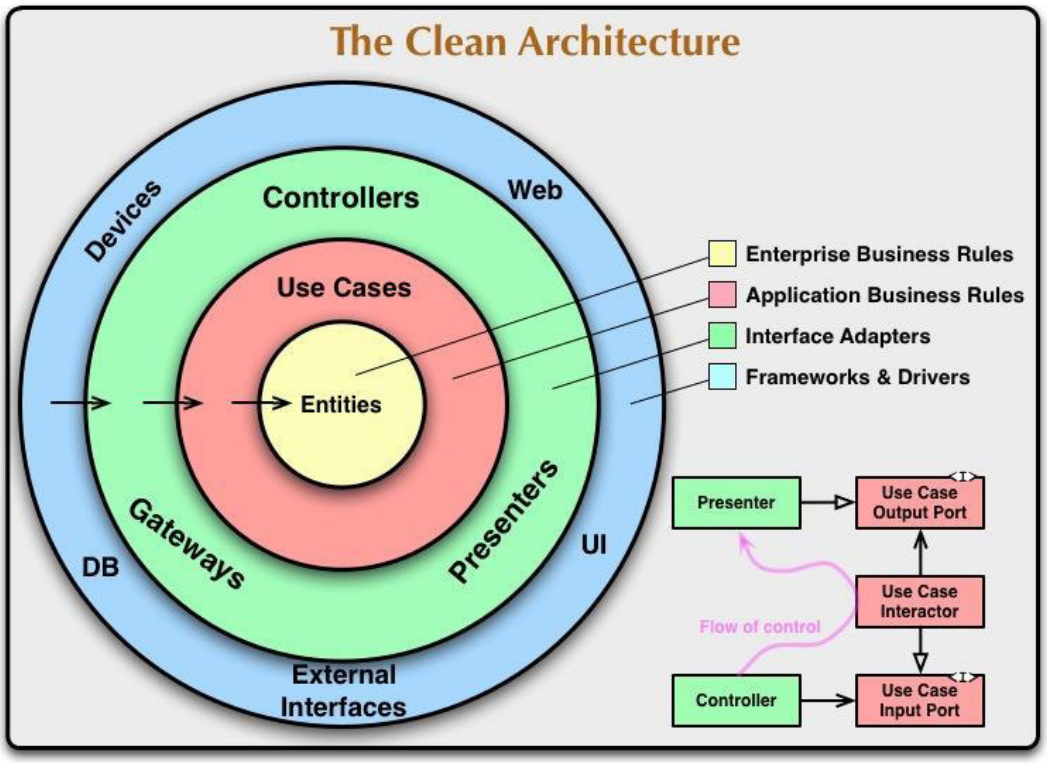


Figura 5 Diagrama Integrado de Clean Architecture

Fuente: Clean Architecture – A Craftsman´s Guide to Software Structure and Design. US: Pearson Education.

En el 2012 Robert C. Martin publica en un blog de tecnología que es "The Clean Architecture" y posteriormente en el 2017 publica el libro "Clean Architecture, guía para especialista en la estructura y el diseño de software", con el objetivo de solucionar una serie de problemas y hacer que el código sea robusto, mantenible, testeable y extensible.

Aunque todas estas arquitecturas varían un poco en los detalles, son muy similares. Todos tienen el mismo objetivo, que es la separación de las preocupaciones. Todos ellos logran la separación dividiendo el software en capas. Cada uno tiene al menos

una capa para las reglas de negocios, y otra capa para las interfaces del usuario y del sistema.

Cada una de las arquitecturas produce sistemas que tienen las siguientes características.

- ✓ Independencia de frameworks: La arquitectura no depende de la existencia de una librería con funciones. Esto le permite utilizar dichos framework como herramientas y no depender de sus limitaciones.

- ✓ Testeable: las reglas de negocio pueden ser probadas in la interface de usuario, base datos, web server u otro elemento exterior.

- ✓ Independiente de la interface de usuario: La interface puede cambiar fácilmente sin cambiar el resto del sistema.

- ✓ Independiente de la Base de Datos: Puedes intercambiar Oracle o SQL Server por Mongo, BigTable, CouchDB o cualquier otro. Tus reglas de negocio no están ligadas a la base de datos.

- ✓ Independiente a cualquier agente externo: las reglas de negocio no conocen nada sobre las interfaces del mundo exterior.

(Martin, Clean Architecture – A Craftsman´s Guide to Software Structure and Design, 2018)

Regla de dependencia:

Los círculos concéntricos en la **Figura 5** representan diferentes áreas de software, la regla primordial que hace que la arquitectura funcione es la regla de dependencia.

Las dependencias de código fuente deben apuntar solo hacia adentro, nada en un círculo interno puede saber nada sobre un círculo exterior. Es decir, el nombre de algo declarado en un círculo externo no debe ser mencionado por el código en un círculo interno, eso incluye funciones, clases, variables o cualquier otra entidad de software

nombrada. Por la misma razón los datos declarados en un círculo externo no deberían ser utilizados por un círculo interno. No deseamos que nada en un círculo externo impacte en los círculos internos. (Martin, Clean Architecture – A Craftsman´s Guide to Software Structure and Design, 2018)

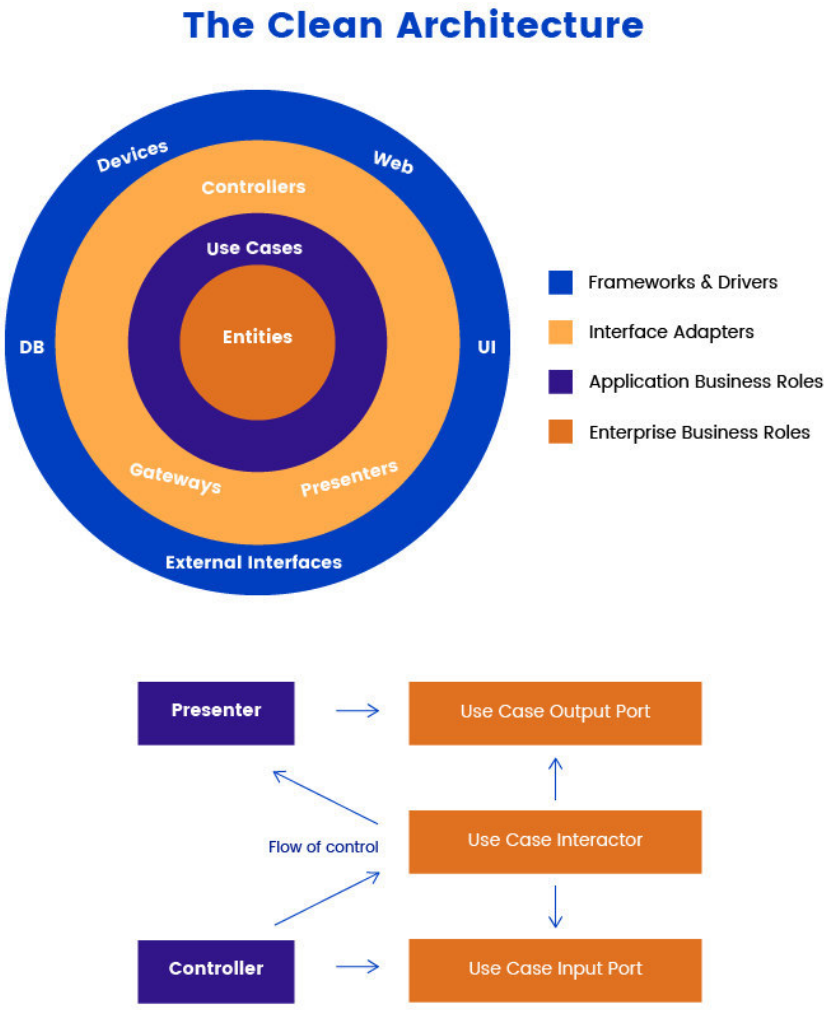


Figura 6. Original Clean Architecture scheme suggested by Robert C. Martin

Fuente: Elaboración propia con datos de Clean Architecture – A Craftsman´s Guide to Software Structure and Design. US: Pearson Education.

- **Entities (Entidades):** Las entidades encapsulan las reglas críticas de negocio. Una entidad puede ser un objeto con métodos, o puede ser un conjunto de estructuras y funciones de datos. No importa siempre que las entidades puedan ser utilizadas por muchas aplicaciones diferentes en la empresa.

- **Use Cases (Casos de uso):** En esta capa se encuentran las reglas de negocio específicas de la aplicación, encapsula e implementa todos los casos de uso del sistema. Los cambios realizados en esta capa no afectan a las entidades, tampoco esperamos que esta capa se vea afectada por las capas externas, como la base de datos, la interfaz de usuario o cualquier otro que se encuentre en una capa superior.

- **Interface Adapter:** esta capa es un adaptador de conjunto que convierte los datos del formato más conveniente para los casos de uso y entidades, también convierte los datos para alguna acción externa, como la base de datos o la web. Es esta capa, por ejemplo, la que contendrá por completo la arquitectura MVC de una GUI. el presentador, las vistas y los controladores pertenecen todos a la capa de adaptador de interfaces. Es probable que los modelos solo sean estructuras de datos que pasan de los controladores a los casos de uso, y que vuelven de los casos de uso al presentador y a la vista.

- **Frameworks and Drivers:** la capa más externa del modelo en la Figura [1] generalmente está compuesta de frameworks y herramientas tales como la UI, Web, base de datos, Devices, etc. Generalmente solo se deben comunicar con el siguiente círculo a su interior.
  (Martin, Clean Architecture – A Craftsman´s Guide to Software Structure and Design, 2018)

3.2.5.8 MVP (Modelo Vista Presentador):

Es un patrón de diseño que notifica a la vista cualquier cambio que sufra el estado del modelo. La información puede pasarse en la propia notificación, o después de la notificación, la vista puede consultar el modelo directamente para obtener los datos actualizados. Por el contrario, en el MVP, la vista no sabe nada sobre el modelo y la función del presentador es la de mediar entre ambos, enlazando los datos con la vista.

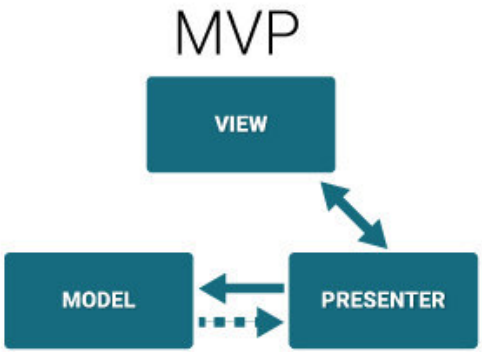


Figura 7. MVP (Patrón de diseño Modelo Vista Presentación)

Fuente: Elaboración propia con datos de https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42

3.2.5.9 Aplicativo Móvil:

Se denomina apps o aplicación al software que se instala en el dispositivo móvil. Se lo llamó como tal desde el inicio del IPhone, la compañía Apple como marketing usó este nuevo nombre para referirse al software que se encuentra subido en las tiendas virtuales, ya sea teléfono o tableta y para su instalación se necesita descargarlo e instalar, algunos son gratuitos y otros tienen costos. Estos se integran a las características del equipo, como su cámara, acelerómetro y sistema de posicionamiento global (GPS), etc.

3.2.5.10 Clean Architecture en Android:

Los principios de Clean Architecture como expone Robert Martin, intentan enfocar al desarrollador en pensar a través de la funcionalidad central de la aplicación. Lo

hace separando la arquitectura de la aplicación en tres capas principales, ver ***Figura 5***, cómo la aplicación muestra los datos al usuario (**capa de presentación**), cuáles son las funciones principales de la aplicación (**dominio o capa de caso de uso**) y cómo pueden ser accedidos los datos (**capa de datos**). La capa de presentación se ubica como la capa más externa, la capa de dominio se ubica en la capa intermedia y la capa de datos reside en la capa interna.

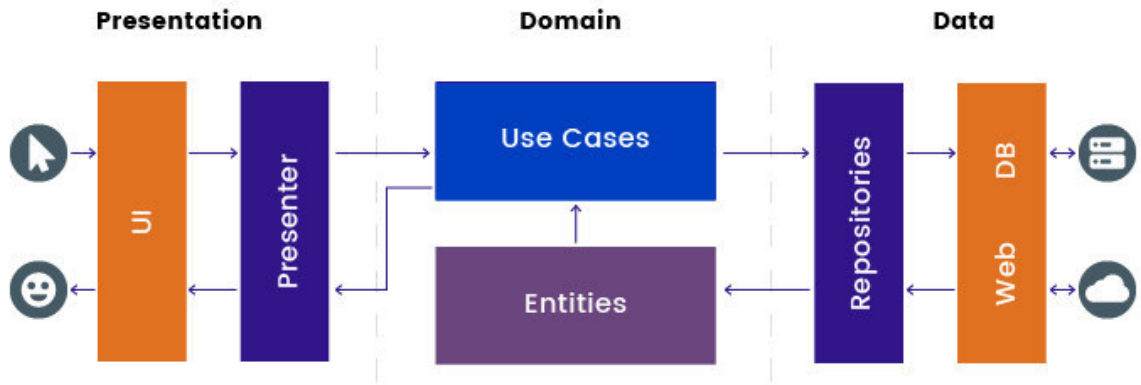


Figura 8. Diagrama de dominio – Clean Architecture

Fuente: Elaboración propia con datos de Clean Architecture – A Craftsman´s Guide to Software Structure and Design. US: Pearson Education.

Es importante tener en cuenta que cada capa tiene su propio modelo de datos, y los datos solo pueden intercambiarse entre capas y, por lo general, solo fluye en una dirección (es decir, de exterior a interna o interna a externa). En cualquier momento es necesario intercambiar datos, generalmente un convertidor utilizado para mapear el modelo de una capa a otra. De esta forma, se crea un límite de separación que ayuda a limitar los cambios en una capa para causar efectos secundarios no deseados en otras capas.

En la capa de datos, cada fuente (es decir, archivo, red, memoria) de datos debe representarse utilizando el patrón de datos del repositorio. Hay muchas formas diferentes de implementar este patrón de repositorio, pero el objetivo final es exponer una interfaz que defina las diferentes consultas que se deben realizar para abstraer el

tipo de fuente de datos utilizada. Por lo tanto, las implementaciones subyacentes pueden intercambiarse independientemente del tipo de fuente de datos utilizada.

La arquitectura limpia introduce más abstracciones e intenta aplicar principios de responsabilidad únicos en el desarrollo de Android. Si bien puede haber inquietudes sobre este enfoque que agrega más complejidad, bajo rendimiento y escasa capacidad de prueba, se ha demostrado que funciona con éxito en aplicaciones de producción. (Architecture of Android Apps, https://guides.codepath.com/android/Architecture-of-Android-Apps, 2018]

3.2.5.11 Android Studio:

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA . Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan tu productividad durante la compilación de apps para Android, como las siguientes:

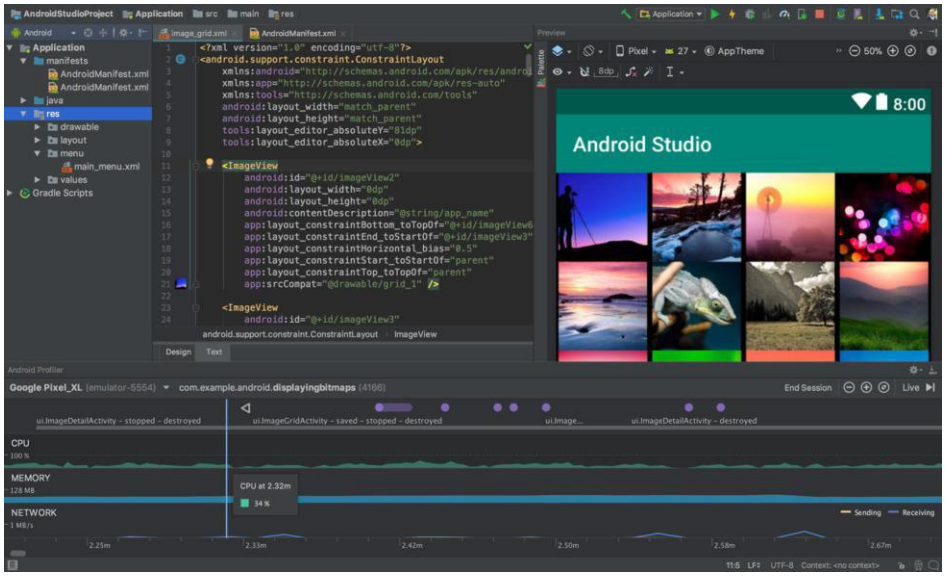


Figura 9 Captura de pantalla de Android Studio

Fuente: Android. (22 de 08 de 2018). Android Developer. Obtenido de developer.android.com: https://developer.android.com/studio/?hl=es-419

- Un sistema de compilación basado en Gradle flexible
- Un emulador rápido con varias funciones
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android
- Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK
- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código
- Gran cantidad de herramientas y frameworks de prueba
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etc.
- Compatibilidad con C++ y NDK
- Soporte incorporado para Google Cloud Platform, lo que facilita la integración de Google Cloud Messaging y App Engine

(Conoce Android Studio, https://developer.android.com/studio/intro/?hl=es-419)

3.2.6 Implementación de Clean Architecture en GMD:

En esta sección se mostrará la arquitectura de software en el desarrollo de aplicaciones Android, a través de diferentes vistas, cada una de las cuales ilustra un aspecto en particular del software como parte de la solución, con la finalidad de presentar una visión global y comprensible del diseño general del software desarrollado, también muestra todo lo necesario para su implementación en nuevos desarrollos.

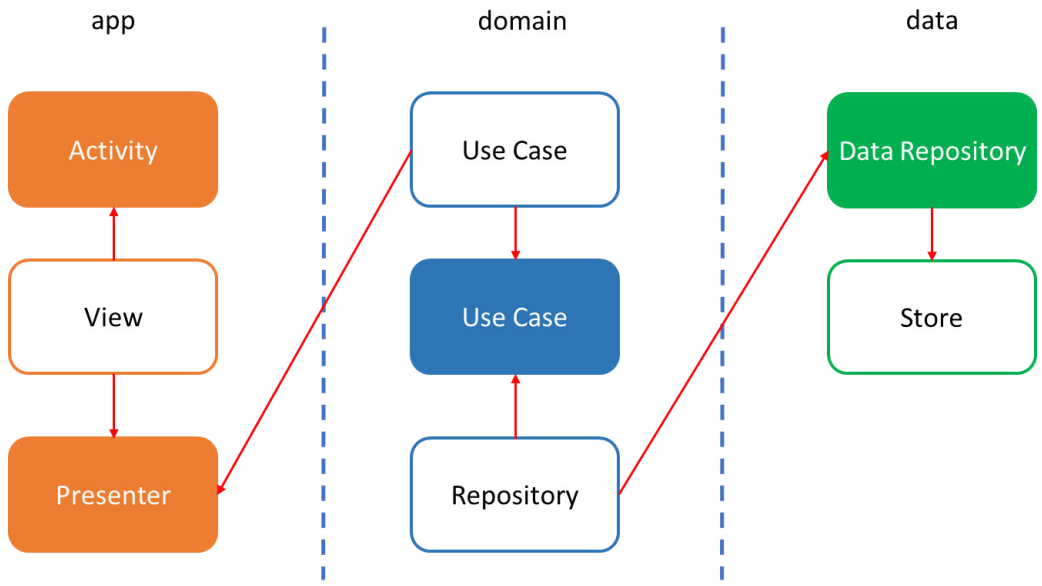


Figura 10 Diagrama Arquitectónico de la solución

Fuente: Elaboración propia, con datos de Guía de implementación de Clean Architecture en GMD.

3.2.6.1 Objetivos y restricciones de arquitectura:

Objetivos de la arquitectura:

- Deberá mostrar independencias de capas.
- Deberá soportar desde la versión 4.0 de Android.
- Deberá tener sus componentes en módulos o librerías independientes con la finalidad de poder reutilizarlos en todos los proyectos en los que se requiera.

Restricciones de la Arquitectura:

- Infraestructura:

- ✓ Desde Android 4.0

- Desarrollo:
  - ✓ Utilizar el framework de inyección de vistas Butter Knife 6.0.0 en la capa de presentación (ver: http://jakewharton.github.io/butterknife/).
  - ✓ Utilizar la librería de Google GSON 2.4 para serializar y deserializar Json (ver: https://github.com/google/gson).
  - ✓ Utilizar la librería de Model Mapper 0.7.5 para mapear entidades y objetos (ver: http://modelmapper.org/).
  - ✓ Utilizar el framework Retrofit 2.2.0 para consumir servicios web en la capa de datos (ver: http://square.github.io/retrofit/).

3.2.6.2 Capas de la Arquitectura:

En esta sección se pasa a descubrir cada una de las capas de la arquitectura definida.

a) Vista Lógica:

El diseño de la arquitectura se encuentra orientado al modelo de Clean Architecture, el cual contiene las siguientes capas y cada capa a su vez contiene los paquetes y clases, a continuación de tallaremos los mismos, ver **Figura 7**.

Capa de Presentación (app), bajo el patrón de modelo, la vista y presentación (MVP):

- **mapper:** Contiene el mapeo y transformación de entidades u objetos de las diferentes capas a modelos de la capa de presentación y viceversa.
- **model:** contiene los modelos que se utilizan en esta capa.
- **navigation:** contiene el navegador de toda la aplicación.
- **presenter:** orquesta la interacción entre la vista y el modelo.
- **service:** implementa los servicios utilizados en la aplicación:
- **util:** contiene los utilitarios correspondientes a la capa.
- **view:** contiene las vistas, estas a si mismo están organizadas en:

- **activity:** contiene las actividades de la aplicación y donde se hace la inyección de vistas.
- **adapter:** contiene los adaptadores de los componentes visuales.
- **widget:** contiene widgets de la aplicación.
- **Interface:** contiene interfaces de las actividades.

Capa de Dominio, contiene los casos de uso, entidades y las interfaces de los repositorios y está estructurado de la siguiente forma:

- **entity:** contiene todas las entidades a utilizar en los casos de uso.
- **exception:** contiene métodos de manejo de excepciones de la capa de dominio.
- **repository:** contiene las interfaces de los repositorios a utilizar para cada caso de uso.
- **usecase:** contiene los casos de uso y las interfaces del mismo.

Capa de Datos, contiene los componentes necesarios para el almacenado de datos locales o por medio de un servicio y está estructurado de la siguiente forma:

- **dao:** contiene los dao para su implementación en la base de datos.
- **entity:** contiene las entidades de la capa de datos que no se encuentren incluidas en la implementación.
- **excepción:** contiene métodos de manejo de excepciones de la capa de datos.
- **net:** contiene los métodos de conexión con los servicios, no se encuentra el consumo de cada servicio.
- **repository:** contiene la implementación de los repositorios de dominio, el cual también se encuentra ordenado por fuente de origen (datasource).
- **util:** contiene utilitarios de la capa de datos.

b) Paquetes de diseño Arquitectónicos:

Los componentes de aplicación están estructurados por un archivo APK que empaqueta los módulos. En la **Figura 10** se muestra la estructura que tendrá cada componente de aplicación

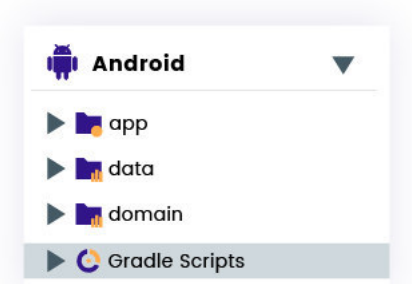


Figura 11. Diagrama de diseño Arquitectónico

Fuentes: Elaboración Propia.

c) Vista de componentes:

Está conformado por una capa de presentación (a**pp)**, una capa de dominio representada por **domain** y de acceso a datos representada por **data**, donde hay que tener en cuenta que el dominio de los componentes está compuesto por el dominio: *pe.gmd.android*. Este componente está a su vez estructurado de la siguiente forma:

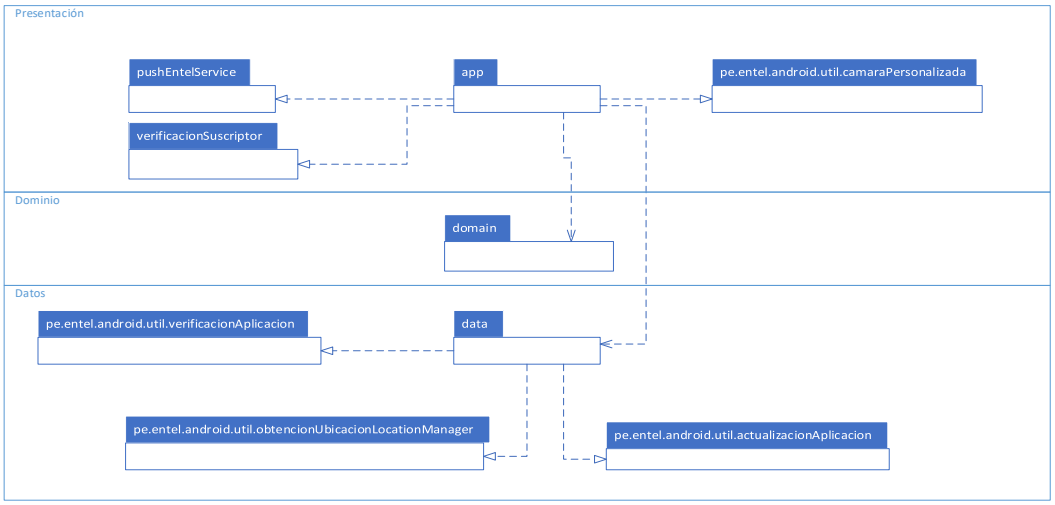


Figura 12 Diagrama de vista de componentes

Fuentes: Elaboración Propia, con datos de guía de implementación de Clean Architecture en GMD

3.2.6.3 Solución en Android:

En esta sección explicaremos en detalle los pasos y especificaciones para implementar Clean Architecture en un proyecto Android usando el IDE (entorno de desarrollo integrado) Android Studio.

Para su implementación se debe de crear 3 módulos con las siguientes características:

Tabla 1 Módulos

Nombre	Capa	Tipo
app	Presentación	Proyecto de Android
domain	Dominio	Módulo de librería java
data	Data	Módulo de librería Android

Fuentes: Elaboración propia

I. Capa de presentación (app)

En la capa de Presentación se debe de implementar cada uno de los siguientes paquetes, en el caso de paquetes funcionales se incluirá un ejemplo y en el caso de arquitectura se pondrá toda la implementación de los objetos:

- App.mapper : contiene final class con métodos estáticos por cada Modelo que se requiera mappear o convertir. No requieren inicializar ni declarar para su uso.

Dentro de los mapper se deben de poner los métodos para mapear entidades a Modelo, Json a Modelo o revertir, en cualquier caso.

```
public final class ProductoModelMapper {


    public static ProductoModel StringAdapter(String text){
        ProductoModel model =  new ProductoModel();

        if(text != null) {
            Gson gson = new Gson();
            model = gson.fromJson(text, ProductoModel.class);
        }
        return model;
    }

    public static ProductoModel EntityAdapter(Producto entidad){
        ProductoModel model =  new ProductoModel();

        if(entidad != null) {
            ModelMapper modelMapper = new ModelMapper();
            model = modelMapper.map(entidad, ProductoModel.class);
        }
        return model;
    }

    public static Producto EntityRevert(ProductoModel model){
        Producto entidad =  new Producto();

        if(model != null) {
            ModelMapper modelMapper = new ModelMapper();
            entidad = modelMapper.map(model, Producto.class);
```

- Entidades a Modelo: en este caso se utiliza la librería de ModelMapper.

```
    public static ProductoModel EntityAdapter(Producto entidad){
        ProductoModel model =  new ProductoModel();

        if(entidad != null) {
            ModelMapper modelMapper = new ModelMapper();
            model = modelMapper.map(entidad, ProductoModel.class);
        }
        return model;
    }
```

- Json a Modelo: en este caso se utiliza la librería del Gson.

```
public static ProductoModel StringAdapter(String text){
    ProductoModel model =  new ProductoModel();

    if(text != null) {
        Gson gson = new Gson();
        model = gson.fromJson(text, ProductoModel.class);
    }
    return model;
}
```

- Revertir: en este caso se utiliza el ModelMapper para convertir el modelo a entidad o Gson para convertir el modelo a Json, según sea el caso.

```
public static Producto EntityRevert(ProductoModel model){
    Producto entidad =  new Producto();

    if(model != null) {
        ModelMapper modelMapper = new ModelMapper();
        entidad = modelMapper.map(model, Producto.class);
```

- App.model: son clases que representa al modelo de la capa presentación:

```
public class ProductoModel {
    public final static String TAG = Producto.class.getSimpleName();;

    private Long id = null;
    private String idProducto;
    private String codigoProducto;
    private String nombre;
    private String precio;
    private String cantidad;
    private String monto;

    public String getIdProducto() { return idProducto; }

    public void setIdProducto(String idProducto) { this.idProducto = idProducto; }

    public String getCodigoProducto() { return codigoProducto; }

    public void setCodigoProducto(String codigoProducto) { this.codigoProducto = codigoProducto; }

    public String getNombre() { return nombre; }
```

```java
public class MenuModel {

    public final static int SINCRONIZAR = 0;
    public final static int SALIR = 1;
    public final static int INICIO = 2;
    public final static int PENDIENTE = 3;
    public final static int FOTO = 4;

    int id;
    String titulo;
    int imagen;

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getTitulo() { return titulo; }

    public void setTitulo(String titulo) { this.titulo = titulo; }

    public int getImagen() { return imagen; }
```

- App.navigation: es una final class, con métodos estáticos que no requiere su inicialización ni implementación, sus métodos deben de recibir el contexto y parámetros a enviar en la navegación:

```java
public final class Navigator {

    public static void navigateToLogin(Context context){
        if(context != null){
            Intent intent = LoginActivity.getCallingIntent(context);
            context.startActivity(intent);
        }
    }

    public static void navigateToFoto(Context context){
        if(context != null){
            Intent intent = FotoActivity.getCallingIntent(context);
            ((Activity)context).startActivityForResult(intent, EntelConfig.CONSIDCAMARA);
        }
    }

    public static void navigateToMensaje(Context context){
        if(context != null){
            Intent intent = MensajeActivity.getCallingIntent(context);
            context.startActivity(intent);
        }
    }

    public static void navigateToMenu(Context context){
        if(context != null){
            Intent intent = MenuActivity.getCallingIntent(context);
            ((Activity)context).finish();
            context.startActivity(intent);
```

- App.presenter : son clases que contienen la interacción de la vista con el modelo, también es el que se comunica con la capa de dominio y presentación:

```
public class LoginPresenter {
    final LoginView view;
    public LoginPresenter(LoginView view) { this.view = view; }

    public void iniciar() {
        obtenerVersion();
        verificarUsuario();
    }

    private void verificarUsuario() {
        UsuarioRepository usuarioRepository = new UsuarioDataRepository(view.getContext());
        GetUsuarioUseCase getUsuarioUseCase = new GetUsuarioUseCaseImpl(usuarioRepository);
        UsuarioModel usuario = UsuarioModelMapper.adapter(getUsuarioUseCase.ejecutar());

        if(usuario!=null) {
            view.actualizarUsuario(usuario.getCodigo(), usuario.getClave());
        }
    }

    private void obtenerVersion() {
```

- App.service: son las clases que contienen los servicios a utilizar en cada aplicación, como las notificaciones y push:

```
public class MensajeNotifier extends IntentService {

    private NotificationManager ioNotifyManager;
    public final static int CODIGO_NOTIFICACION_ESTADO = 133707736;

    public MensajeNotifier() { super("MensajeNotifier"); }

    @Override
    public void onCreate() {
        super.onCreate();
        ioNotifyManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    }

    @Override
    protected void onHandleIntent(Intent intent) {

        String lsMensaje = intent.getStringExtra(EntelConfig.PUSH_BUNDLE_MENSAJE).substring(1);
        String lsCodMensaje = intent.getStringExtra(EntelConfig.PUSH_BUNDLE_CODMENSAJE);
        showNotificacion(lsMensaje, true);
    }

    public void showNotificacion(String mensaje, boolean flgSonido) {
        Intent loIntent = new Intent(this, MensajeActivity.class);
        loIntent.putExtra(EntelConfig.PUSH_BUNDLE_MENSAJE,mensaje);

        int liCodigoNotificacion = CODIGO_NOTIFICACION_ESTADO;
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, loIntent,
                //Intent.FLAG_ACTIVITY_REORDER_TO_FRONT
                // |
                PendingIntent.FLAG_UPDATE_CURRENT
                        | PendingIntent.FLAG_ONE_SHOT
        );
```

```
public class PushBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context poContext, Intent poIntent) {
        Bundle loBundle = poIntent.getExtras();

        String topic = loBundle.getString(EntelConfig.PUSH_BUNDLE_TOPICNAME);
        String topicValidacion = UtilitarioApp.fnReadParamProperty(poContext, "MQTT_TOPIC

        Log.v("PUSH", "topic; " + topic);
        Log.v("PUSH", "topicValidacion; " + topicValidacion);

        if (topicValidacion.equals(topic)) {
            String codigo = loBundle.getString(EntelConfig.PUSH_BUNDLE_CODMENSAJE);
            String mensaje = loBundle.getString(EntelConfig.PUSH_BUNDLE_MENSAJE);
            Log.v("PUSH", "codigo; " + codigo);
            Log.v("PUSH", "mensaje; " + mensaje);

            Intent loIntent = new Intent(poContext, MensajeNotifier.class);
            loIntent.putExtra(EntelConfig.PUSH_BUNDLE_MENSAJE,mensaje);
            loIntent.putExtra(EntelConfig.PUSH_BUNDLE_CODMENSAJE,codigo);
            poContext.startService(loIntent);

        }

    }
}
```

- App.util: se encuentran los utilitarios de la aplicación:
  - UtilitarioApp

```
public class UtilitarioApp {

    private static final String TAG = "UtilitarioApp";

    public static String CONSPROPERTYFILE = "params.properties";

    public static String fnVersion(Context poContext) {
        String lsVersion = "";
        PackageInfo loPackageInfo = null;
        try {
            loPackageInfo = poContext.getPackageManager().getPackageInfo(
                    poContext.getPackageName(), 0);
        } catch (PackageManager.NameNotFoundException e) {
            e.printStackTrace();
            Log.v(TAG, "Verificar el manifest: " + e.getMessage());
            return "ERROR VERSION";
        }

        lsVersion = loPackageInfo.versionName;

        return lsVersion;
    }

    public static Properties readProperties(Context ctx) {
        return readProperties(ctx, EntelConfig.CONSPROPERTYFILE);
    }
```

```
public static Properties readProperties(Context ctx, String arc) {
    Properties prop = null;
    try {
        AssetManager am = ctx.getAssets();
        InputStream is = am.open(arc);
        prop = new Properties();
        prop.load(is);
    } catch (Exception ex) {
        Log.e(TAG, "readProperties: ", ex);
    }
    return prop;
}
```

- App.view: se encuentran en esta raíz todas las interfaces de las actividades, ver **Figura 12**.

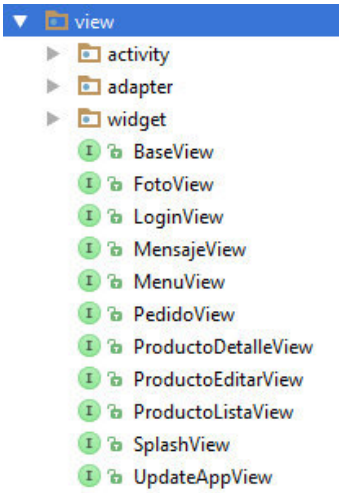


Figura 13 Diagrama del paquete App.view

Fuente:Elaboración propia, captura de pantalla android Studio

```
public interface FotoView extends BaseView{

    void mostrarPreview();

    void hideCamera();

    void preview(Bitmap bMapRotate);

    void saveFotoOk();

    void mostrarEnvio();

    void saveFotoError();

    void showCamera();
}
```

- App.view.activity: se encuentran todas las actividades que implementan las interfaces descritas en el punto anterior, también se realiza la inyección de vistas:

```
public class FotoActivity extends BaseActivity implements FotoView , CameraFragment.PerformerCallback, CameraFragment.TipoCamaraCa

    FotoPresenter presenter;

    @InjectView(R.id.toma_foto_ib_enviar)
    ImageButton ibEnviarFoto;
    @InjectView(R.id.toma_foto_iv_preview)
    ImageView ivPreview;

    CameraFragment cameraFragment;

    private boolean grabada = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_foto);
        ButterKnife.inject(this);
        cameraFragment = (CameraFragment)getSupportFragmentManager().findFragmentById(R.id.toma_foto_camera_fragment);
        presenter = new FotoPresenter(this);
    }

    @Override
```

- App.view.adapter: se encuentran todos los adaptadores de los componentes visuales.

```
public class MenuPrincipalAdapter extends ArrayAdapter {

    int resource;


    public MenuPrincipalAdapter(Context context, int resource, List<MenuModel> list) {
        super(context, resource, list);
        this.resource = resource;

    }

    public View getView(int position, View convertView, ViewGroup parent) {
        LinearLayout nuevaVista;
        LayoutInflater inflater = (LayoutInflater) getContext()
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        if (convertView == null) {
            nuevaVista = new LinearLayout(getContext());
            inflater.inflate(resource, nuevaVista, true);

        } else {
            nuevaVista = (LinearLayout) convertView;
        }
```

- App.view.widget : se encuentran todos los widget de la aplicación.

```
public class AppWidget extends AppWidgetProvider {

    static void updateAppWidget(Context context, AppWidgetManager appWidgetManager,
                                int appWidgetId) {

        CharSequence widgetText = "EXAMPLE";
        RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.app_widget);
        views.setTextViewText(R.id.appwidget_text, widgetText);

        appWidgetManager.updateAppWidget(appWidgetId, views);
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        for (int appWidgetId : appWidgetIds) {
            updateAppWidget(context, appWidgetManager, appWidgetId);
        }
    }
```

II. Capa de dominio (domain)

En la capa de Dominio se debe de implementar cada uno de los siguientes paquetes, en el caso de paquetes funcionales se incluirá un ejemplo y en el caso de arquitectura se pondrá toda la implementación de los objetos.

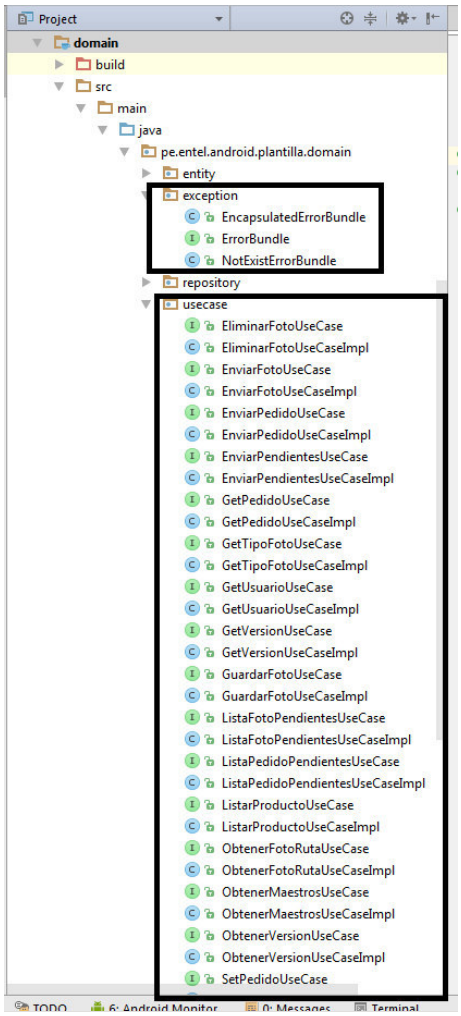


Figura 14 Diagrama del paquete de la capa de dominio

Fuente:Elaboración propia, captura de pantalla android Studio

- Domain.entity : se encuentran todas la entidades del dominio

```
public class Localizacion {

    private String latitud;
    private String longitud;
    private String gpsPrecision;
    private String velocidad;
    private String fechaMovil;
    private String gpsOrigen;


    public String getLatitud() { return latitud; }

    public void setLatitud(String latitud) { this.latitud = latitud; }

    public String getLongitud() { return longitud; }

    public void setLongitud(String longitud) { this.longitud = longitud; }

    public String getVelocidad() { return velocidad; }

    public void setVelocidad(String velocidad) { this.velocidad = velocidad; }
```

- Domain.exception: contiene los métodos de manejo de excepciones de la capa de Dominio:

  - EncapsulatedErrorBundle

```
public class EncapsulatedErrorBundle implements ErrorBundle {

    private Exception error;

    public EncapsulatedErrorBundle(Exception e) { this.error = e; }

    @Override
    public Exception getException() { return error; }

    @Override
    public String getErrorMessage() {
        if (error != null)
            return error.getMessage();
        else
            return "No definido";
    }
}
```

  - ErrorBundle

```
public interface ErrorBundle {
    Exception getException();

    String getErrorMessage();
}
```

- NotExistErrorBundle

```
public class NotExistErrorBundle implements ErrorBundle {

    private String nombre;

    public NotExistErrorBundle(String nombre) { this.nombre = nombre; }

    @Override
    public Exception getException() { return new Exception("Not Exist " + nombre); }

    @Override
    public String getErrorMessage() { return "Not exist " + nombre; }
}
```

- Domain.repository: contiene las interfaces de los repositorios a implementar en la capa de datos:

```
public interface FotoRepository {

    void guardarFoto(String idUsuario, String identificador, byte[] data, GuardarFotoCallback guardarFotoCallback);

    void enviarFoto(String identificador, EnviarFotoCallback enviarFotoCallback);

    void eliminarFoto(String identificador);

    String obtenerRutaFoto(String identificador);

    int getTipoFoto();

    void setTipoFoto(int tipo);

    List<Photo> listarFotosPendientes();

    interface GuardarFotoCallback {
        void onGuardado();
        void onError(ErrorBundle errorBundle);
    }

    interface EnviarFotoCallback{
```

- Domain.usecase : contiene todas las interfaces de los casos de uso:

```
public interface ListaPedidoPendientesUseCase {
    List<Pedido> ejecutar();
}
```

- Domain.usecase.implementacion : contiene la implementación de los caso de uso:

```
public class EnviarPedidoUseCaseImpl implements EnviarPedidoUseCase {

    final PedidoRepository pedidoRepository;
    final LocationRepository locationRepository;
    final UsuarioRepository usuarioRepository;

    public EnviarPedidoUseCaseImpl(PedidoRepository pedidoRepository, LocationRepository locationRepository, UsuarioRepository u
        this.pedidoRepository = pedidoRepository;
        this.locationRepository = locationRepository;
        this.usuarioRepository = usuarioRepository;
    }

    @Override
    public void ejecutar(final Pedido pedido, final Callback callback) {

        locationRepository.obtenerLocalizacion((localizacion) -> {
                enviarPedido(pedido,localizacion,callback);
        });

    }
```

III. Capa de Data (data)

En la capa de Data se debe de implementar cada uno de los siguientes paquetes, en el caso de paquetes funcionales se incluirá un ejemplo y en el caso de arquitectura se pondrá toda la implementación de los objetos:

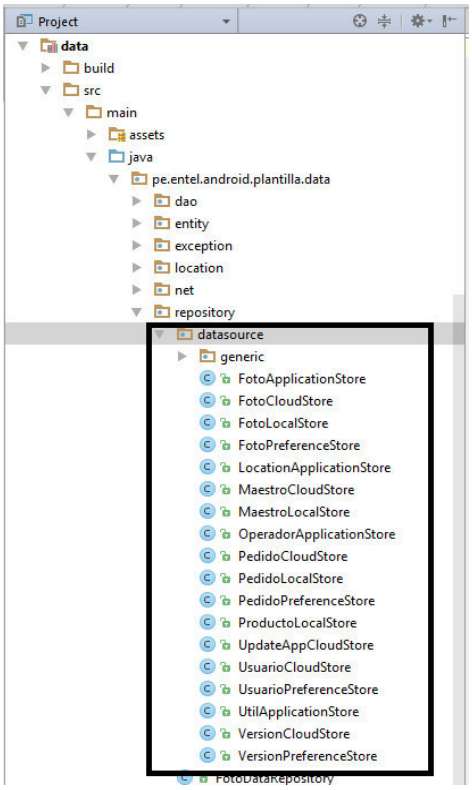


Figura 15 Diagrama del paquete de la capa de data

Fuente:Elaboración propia, captura de pantalla android Studio

- Data.dao: contiene las entidades a utilizar en la implementación del Sugar:

```
public class FotoDao extends SugarRecord {

    private String idUsuario;
    private String identificador;
    private String flgEnviado;

    public String getIdentificador() { return identificador; }

    public void setIdentificador(String identificador) { this.identificador = identificador; }

    public String getIdUsuario() { return idUsuario; }

    public void setIdUsuario(String idUsuario) { this.idUsuario = idUsuario; }

    public String getFlgEnviado() { return flgEnviado; }

    public void setFlgEnviado(String flgEnviado) { this.flgEnviado = flgEnviado; }
}
```

- Data.entity: contiene las entidades que no se utilizan en la implementación del Sugar:

```
public class FotoEntity extends PhotoEntity{

    private static final String TAG = FotoEntity.class.getName();

    private String idUsuario;
    private byte[] imagen;

    public String getIdUsuario() { return idUsuario; }

    public void setIdUsuario(String idUsuario) { this.idUsuario = idUsuario; }

    @Override
    public void agregarDatos(DataOutputStream outputStream) throws IOException {
        Log.v(TAG,"identificador"+this.identificador);
        Log.v(TAG,"idUsuario"+this.idUsuario);
        outputStream.writeLong(Long.parseLong(this.identificador));
        outputStream.writeInt(Integer.parseInt(this.idUsuario));
    }


}
```

- Data.exception: contiene los métodos de manejo de excepciones de la capa de Datos:

    - NetworkConnectionException

```
public class NetworkConnectionException extends Exception {

    public NetworkConnectionException() { super(); }

    public NetworkConnectionException(final String message) { super(message); }

    public NetworkConnectionException(final String message, final Throwable cause) {
        super(message, cause);
    }

    public NetworkConnectionException(final Throwable cause) { super(cause); }
}
```

    - RepositoryErrorBundle

```
public class RepositoryErrorBundle implements ErrorBundle {

    private final Exception exception;

    public RepositoryErrorBundle(Exception exception) { this.exception = exception; }

    public Exception getException() { return exception; }

    public String getErrorMessage() {
        String message = "";
        if (this.exception != null) {
            message = this.exception.getMessage();
        }
        return message;
    }
}
```

    - UserNotFoundException

```
public class UserNotFoundException extends Exception {

    public UserNotFoundException() { super(); }

    public UserNotFoundException(final String message) { super(message); }

    public UserNotFoundException(final String message, final Throwable cause) {
        super(message, cause);
    }

    public UserNotFoundException(final Throwable cause) { super(cause); }
}
```

- Data.repository: contiene la implementación de los repositorios, descritos en la capa de dominio:

```
public class FotoDataRepository implements FotoRepository {

    final Context context;

    public FotoDataRepository(Context context) { this.context = context; }

    @Override
    public void guardarFoto(final String idUsuario, final String identificador, byte[] data, final GuardarFotoCallback guardar
        FileStore fileStore = new FileStore(context);
        FotoApplicationStore applicationStore = new FotoApplicationStore(context, fileStore);
        final FotoLocalStore fotoLocalStore = new FotoLocalStore();

        applicationStore.guardarFoto(identificador, data, new FotoApplicationStore.GuardarFotoCallback() {
            @Override
            public void onGuardado() {
                fotoLocalStore.guardarFoto(idUsuario,identificador);
                guardarFotoCallback.onGuardado();
            }

            @Override
            public void onError(Exception e) {
                guardarFotoCallback.onError(new RepositoryErrorBundle(e));
            }
        });
    }
```

- Data.repository.datasource.application: contiene los métodos para registrar en el dispositivo móvil.

```
public class FotoApplicationStore {

    final Context context;
    final FileStore fileStore;
    final String raiz;

    public FotoApplicationStore(Context context, FileStore fileStore) {
        this.context = context;
        this.fileStore = fileStore;
        raiz = UtilitarioData.getRaiz(context);
    }

    public void guardarFoto(String identificador, byte[] data, GuardarFotoCallback guardarFotoCallback ){

        try {
            fileStore.storeFileImage(data,raiz,identificador);
        } catch (Exception e) {
            guardarFotoCallback.onError(e);
        }

        guardarFotoCallback.onGuardado();
    }

    public String obtenerFotoRuta(String identificador){
        return fileStore.fnRutGrabado(raiz,identificador);
    }
```

- Data.repository.datasource.cloud: contiene los métodos para conectarse con los servicios web.

```
public class FotoCloudStore {

    final Context context;
    EnviarFotoCallback enviarFotoCallback;


    public FotoCloudStore(Context context) { this.context = context; }


    public void enviarFoto(FotoEntity fotoEntity, final EnviarFotoCallback enviarFotoCallback ){
        this.enviarFotoCallback = enviarFotoCallback;

        if (UtilitarioData.isThereInternetConnection(context)) {
            Gson gson = new Gson();

            Intent intent = new Intent(context, BinaryService.class);
            intent.setData(Uri.parse(Url.obtenerRuta(context,
                    Url.EnumUrl.FOTO)));
            Log.v("URL", Url.obtenerRuta(context, Url.EnumUrl.FOTO));
            intent.putExtra(BinaryService.EXTRA_ONLY_A_PARAM, gson.toJson(fotoEntity));
            intent.putExtra(BinaryService.EXTRA_RESULT_RECEIVER, enviarFoto.getResultReceiver());
            context.startService(intent);
        } else {
            enviarFotoCallback.onError(new NetworkConnectionException("Fuera de cobertura"));
        }

    }
```

- Data.repository.datasource.local: contiene los métodos para registrar en la base de datos de la aplicación.

```
public class FotoLocalStore {

    public void guardarFoto(String idUsuario, String identificador){
        FotoDao fotoDao = new FotoDao();
        fotoDao.setIdUsuario(idUsuario);
        fotoDao.setIdentificador(identificador);
        fotoDao.setFlgEnviado("F");
        fotoDao.save();
    }

    public void borrarFoto(String identificador){
        FotoDao.deleteAll(FotoDao.class,"identificador = ?",identificador);
    }

    public FotoDao obtenerFoto(String identificador){
        return FotoDao.find(FotoDao.class,"identificador = ?",identificador).get(0);
    }

    public void guardarFoto(FotoDao fotoDao) { fotoDao.save(); }

    public List<FotoDao> listarFotosPendientes(){
        return FotoDao.find(FotoDao.class,"flg_enviado = 'F'");
    }

}
```

- Data.repository.datasource.preference: contiene los métodos para registrar en las preferencias de la aplicación.

```
public class FotoPreferenceStore extends PreferenceStore {

    private final static String TIPO_CAMARA ="TIPO_CAMARA";

    public FotoPreferenceStore(Context context) { super(context); }

    public void setTipoCamara(int tipo) {
        saveOnSharePreferences(TIPO_CAMARA,tipo);

    }

    public int getTipoCamara() { return getPreferenceInt(TIPO_CAMARA); }
}
```

IV. Conexión de capas:

En el paquete presenter de la capa de Presentación, se definen cada una de las interacciones de la vista, modelo, dominio y datos.

En el caso de conectar la capa de Presentación a la capa de Datos para enviar a su respectivo caso de uso se debe de hacer lo siguiente:

```
UsuarioRepository usuarioRepository = new UsuarioDataRepository(view.getContext());
```

Donde UsuarioRepository representa la interface del repositorio descrita en el paquete domain.repository, siendo esta implementada por UsuarioDataRepositorio, el cual se encuentra en el paquete data.repository, la implementación suele recibir el contexto de la aplicación para su uso en la capa de datos.

En el caso de conectar la capa de Presentación con la capa de Dominio para ejecutar sus casos de uso se debe de hacer lo siguiente:

Donde GuardarFotoUseCase representa la interface del caso de uso descrita en el paquete domain.usecase, siendo esta implementada por GuardarFotoUseCaseImpl, el cual se encuentra en el paquete domain.usecase.implementation, la implementación suele recibir las interfaces de los repositorios previamente instanciados.

V. Uso de Retorfit:

En el proyecto de la capa data se debe de incluir la dependencia del Retrofit dentro del build.gradle:

```
compile 'com.squareup.retrofit2:retrofit:2.2.0'
compile 'com.squareup.retrofit2:converter-gson:2.2.0'
```

Se debe de iniciar el Retrofit en los datasource de data.repository.datasource.cloud:

Para mayor información de cada uno de los métodos: http://square.github.io/retrofit.

3.3 Evaluación:

3.3.1 Evaluación económica

La evaluación económica incluye lo siguiente:

- Costos del Proyecto
- Personal requerido por mes
- Flujo de pagos
- Flujo de caja
- VAN, TIR

Duración del proyecto: 3 meses

Mes 1 - Costo de personal:

Tabla 2. Costo de personal (Mes1)

Personal Requerido	HH por mes	Costo por HH	# de personas	Costo mensual
Analista programador I	80	S/.30.00	1	S/.2,400.00
Analista programador II	40	S/.30.00	3	S/.3,600.00
Scrum Master	80	S/.35.00	1	S/.2,800.00

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Mes 2 - Costo de personal:

Tabla 3. Costo de personal (Mes 2)

Personal Requerido	HH por mes	Costo por HH	# de personas	Costo mensual
Analista programador I	80	S/.30.00	1	S/.2,400.00
Analista programador II	40	S/.30.00	3	S/.3,600.00
Scrum Master	80	S/.35.00	1	S/.2,800.00

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Mes 3 - Costo de personal:

Tabla 4. Costo de personal (Mes 3)

Personal Requerido	HH por mes	Costo por HH	# de personas	Costo mensual
Analista programador I	80	S/.30.00	1	S/.2,400.00
Analista programador II	0	S/.30.00	3	0
Scrum Master	80	S/.35.00	1	S/.2,400.00

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Costo Hardware:

Tabla 5. Costo de Hardware

Hardware	**Cantidad**	**precio**	**Costo mensual**
Laptop Core i7	1	S/. 150	S/. 150
Laptop Core i5	4	S/. 100	S/. 400
2 equipos móviles Android	2	S/. 80	S/. 160

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Otros costos:

Tabla 6. Otros costos

Otros costos	**Costo mensual**
Luz	S/. 75
Agua	S/. 50
Mantenimiento instalaciones	S/. 200

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Flujo de Pagos:

Tabla 7. Flujo de Pagos

	Oct-17	Nov-17	Dic-17
Analista programador I	S/. 2,400.00	S/.2,400.00	S/.2,400.00
Analistas programadores II	S/. 3,600.00	S/.3,600.00	S/. -
Scrum Master	S/. 2,800.00	S/.2,800.00	S/.2,800.00
Laptop Core i7	S/. 150.00	S/. 150.00	S/. 150.00
Laptop Core i5	S/. 400.00	S/. 400.00	S/. 100.00
2 equipos móviles Android	S/. 160.00	S/. 160.00	S/. 160.00
	S/. 9,510.00	**S/.9,510.00**	**S/.5,610.00**

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Tabla 8. Flujo de Caja

	Oct-17	Nov-17	Dic-17	Ene-18	Feb-18	Mar-18	Abr-18	May-18
Flujo de Ingreso	S/. -	S/. -	S/. -	S/.1,500.00	S/. 2,500.00	S/. 4,000.00	S/. 6,000.00	S/. 7,000.00
Flujo de egreso	S/. 9,510.00	S/.9,510.00	S/.5,610.00	S/. -	S/. -	S/. -	S/. -	S/. -
Flujo de caja	**-S/. 9,510.00**	**-S/.9,510.00**	**-S/.5,610.00**	**S/.1,500.00**	**S/. 2,500.00**	**S/. 4,000.00**	**S/. 6,000.00**	**S/. 7,000.00**
	Jun-18	**Jul-18**	**Ago-18**	**Set-18**	**Oct-18**	**Nov-18**	**Dic-18**	
Flujo de Ingreso	S/. 8,000.00	S/. 10,000.00	S/. 15,000.00	S/. 17,000.00	S/. 20,000.00	S/. 22,000.00	S/. 25,000.00	
Flujo de egreso	S/. -	S/. -	S/. -	S/. -	S/. -	S/. -	S/. -	
Flujo de caja	**S/. 8,000.00**	**S/. 10,000.00**	**S/. 15,000.00**	**S/. 17,000.00**	**S/. 20,000.00**	**S/. 22,000.00**	**S/. 25,000.00**	

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

Flujo de Pagos:

VAN: Valor actual neto

TIR: Tasa interna de retorno

Tasa de descuento: 11%

Tabla 9. VAN - TIR

VAN	S/.21,896.71
TIR	21%

Fuente:Elaboración propia, con datos de evaluación económica de proyectos moviles en el área de innovación de GMD

3.3.2 Interpretación del VAR y TIR:

La empresa trata de lograr un VAN positivo en este caso a través de la implementación de Clean Architecture en los proyectos que tiene a cargo.

Asumiendo una tasa de descuento del 12%, que usualmente es el porcentaje de rentabilidad que la organización considera como mínimo para los proyectos se aplica la fórmula del VAN para un periodo de doce meses que se considera como plaza para poder generar flijos de caja que permitan añadir valor a la empresa luego de la implementación de Clean Architecture.

Dado que la TIR es el retorno efectivo que entregan los flujos de caja proyectados, esa tasa de retorno puede ser comparada luego con la tasa de descuento de la empresa, que vendría a ser la tasa de retorno mínima que debe alcanzar un proyecto de inversión para una compañía. Para este proyecto la TIR supera en bastante a la tasa de descuento de la empresa, lo cual daba una señal de poder seguir adelante con el proyecto, más aún que se respaldaba en los lineamientos de la empresa.

CAPITULO IV. REFLEXIÓN CRÍTICA DE LA EXPERIENCIA:

La implementación de Clean Architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD, ha permitido en el transcurso del ciclo de vida del software puedan incrementar funcionalidad, cambiar de origen de datos, rediseñar las interfaces, actualizar las librerías externas y realizar pruebas en distintos entornos (desarrollo, producción); sin tener mayores dificultades y reduciendo los costos a largo plazo, Por lo que se puedo concretar en el cumplimiento exitoso de los proyectos, evidenciando eficiencia, eficacia y oportunidad de mejora en nuevos proyectos.

4.1 Aporte en el área de desarrollo y responsabilidades

La participación del autor del presente informe en el proceso de implementación de Clean Architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD, fue de Analista Programador y mis funciones específicas fueron elaboración de propuestas de arquitectura para proyectos Android y desarrollo del aplicativo en Android que implementará Clean Architecture, el cual tenga por finalidad servir de plantilla base de los proyectos Android de GMD.

El equipo estuvo formado por 1 scrum master y 4 Analistas programadores, el scrum master fue el encargado gestionar el proyecto y coordinar las reuniones semanales de evaluación de arquitectura, esto para exponer, analizar y reflexionar sobre las propuestas e ideas expuestas de los integrantes del equipo. Luego de realizar la fase evaluación de arquitectura, se tuvo que elegir una que cumpla con las necesidades de GMD y esta fue Clean Architecture, posteriormente se tuvo que desarrollar un proyecto que implemente Clean Architecture, este desarrollo fue realizado por mí persona, el cual se detalla en la sección 3.2.6.3, por último, los resultados obtenidos fueron validados, documentados y comunicados a toda el área de Innovación de GMD.

CAPITULO V. CONCLUSIONES Y RECOMENDACIONES:

5.1 Conclusiones

El uso de Clean Architecture en los proyectos Android de GMD fue adecuadamente implementado, y sirvió para mejorar el desarrollo de los aplicativos en todo el ciclo de vida de software.

I. Se evaluó las arquitecturas en Android y valido que cumplan con las necesidades de GMD, Clean Architecture demostró los proyectos desarrollados bajos esta arquitectura son más escalables, mantenibles y fáciles de realizar pruebas.

II. Se eligió la arquitectura para Android. Mediante una serie de reuniones y exposiciones, donde se llegó a la conclusión que la mejor opción era Clean Architecture, porque cumplió con las necesidades de GMD.

III. Se codificó un aplicativo Android con el IDE Android Studio. El proyecto implemento 3 capas básicas (app, dominio y data) vinculadas y dentro de cada capa los paquetes.

IV. Se validó, luego del desarrollo se evaluó el aplicativo con el equipo y se realizaron pruebas.

V. Se documentó y comunicó el resultado de este proyecto a los integrantes del área de innovación de GMD.

5.2 Recomendaciones

- ✓ Es necesario que el personal de la empresa tenga buena disponibilidad para aceptar y contribuir con nuevas propuestas de mejoras en la empresa, por esta razón es necesario que se cuente con un staff de profesionales que se encuentre muy bien capacitados y actualizados.

- ✓ Clean Architecture es una solución muy buena, pero costosa al inicio, la curva de aprendizaje es alta y requiere experiencia previa de patrones de diseño.

- ✓ Clean Architecture es una arquitectura robusta y compleja, por ello no es recomendable en proyectos cortos o sencillos.

- ✓ Tanto el personal técnico como administrativo deben de comprometerse con la mejora continua en todos los procesos de la empresa, es aquí donde el apoyo de la alta dirección es necesario.

- ✓ Usar patrones de diseño nos ayudan a programar de mejor y cumplir muchos de los principios o reglas de diseño.

- ✓ La inyección de dependencias en la vista ayuda a tener un código más limpio y ordenado.

5.3 Fuentes de información:

- [1] "Clean Architecture – A Craftsman´s Guide to Software Structure and Design", Robert C. Martin; 2018
- [2] "Clean Code – A Handbook of Agile Software Craftsmanship", Robert C. Martin; 2009

5.4 Glosario

- APK: Un archivo con extensión (.apk) es un paquete para el sistema operativo Android.

- CMMI: Integración de modelos de madurez de capacidades o Capability Maturity Model Integration (CMMI) es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

- Driver: Un driver o controlador de dispositivo es el software que comunica los periféricos con el sistema operativo.

- Framework: entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

- IEEE: El Instituto de Ingeniería Eléctrica y Electrónica es una asociación mundial de ingenieros dedicada a la estandarización y el desarrollo en áreas técnicas.

- IDE: en inglés Integrated Development Environment, Entorno de desarrollo integrado, es una aplicación informática que proporciona servicios integrales para facilitarle el desarrollo.

- MVC: es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

- MVP: Es un patrón de diseño que notifica a la vista cualquier cambio que sufra el estado del modelo. La información puede pasarse en la propia notificación, o después de la notificación, la vista puede consultar el modelo directamente para obtener los datos actualizados. Por el contrario, en el MVP, la vista no sabe nada sobre el modelo y la función del presentador es la de mediar entre ambos, enlazando los datos con la vista.

- Scrum: Es una estructura marco de trabajo para completar proyectos, sean simples o complejos, tan exitosamente como sea posible, considerando **principalmente** los requerimientos del cliente. Scrum fue originalmente concebido para proyectos de desarrollo de software, pero se ha visto que funciona efectivamente para cualquier enfoque de trabajo, sea rutinario y simple, o innovador y complejo.

- Gradle: define configuraciones de compilación que se aplican a todos los módulos de tu proyecto.

- Sql: es un lenguaje específico del dominio utilizado en programación, diseñado para administrar sistemas de gestión de bases de datos relacionales

- Inyección de dependencias: es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos

ANEXOS

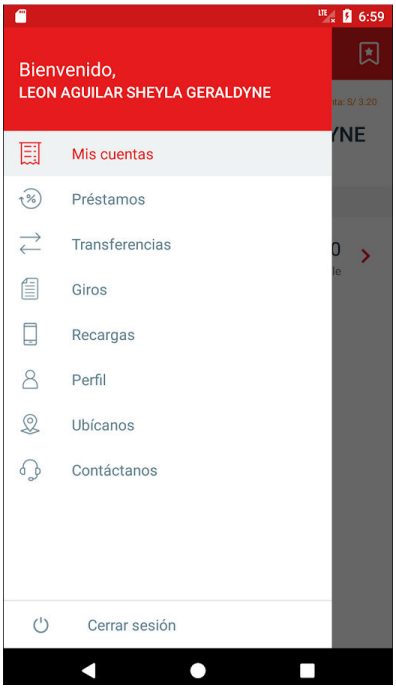
Aplicaciones móviles Android que incorporaron Clean Architecture en GMD.

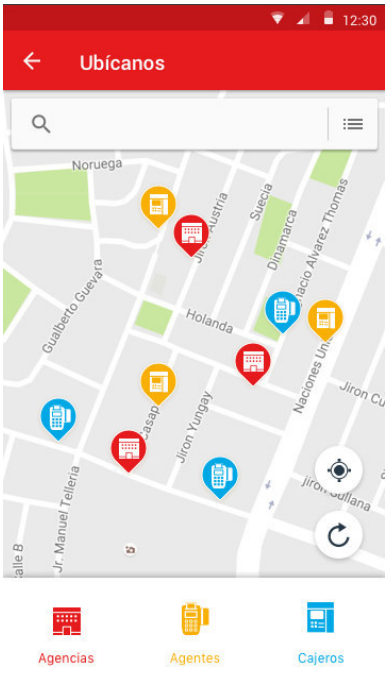
A continuación, se describirá algunos aplicativo que

Banco de la Nación – Banca Móvil

Con la aplicación del Banco de la Nación puedes realizar las siguientes operaciones en cualquier momento y lugar:

- Consulta de saldos y últimos movimientos de tu cuenta de ahorros
- Transferencias mismo banco
- Emisión de giros
- Recargas Claro y Movistar
- Consulta y renovación de préstamos en línea
- Además, podrás ubicar nuestros canales más cercanos en nuestra sección “Ubícanos”





Págalo.pe:

Págalo.pe es una plataforma digital diseñada para simplificar el pago de tasas de diferentes entidades públicas, sin necesidad de ir a una agencia del Banco de la Nación.

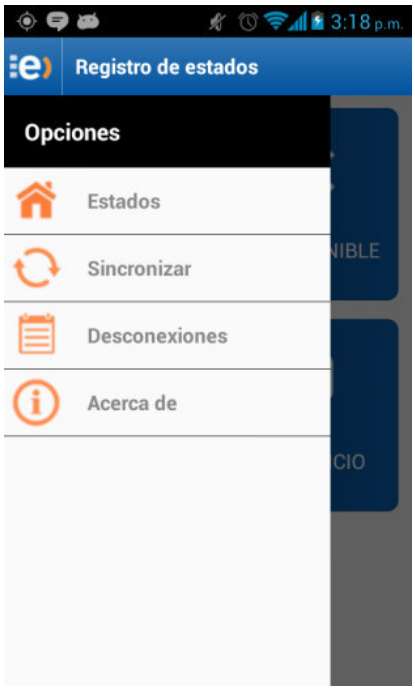
¿Qué pagos se pueden realizar?

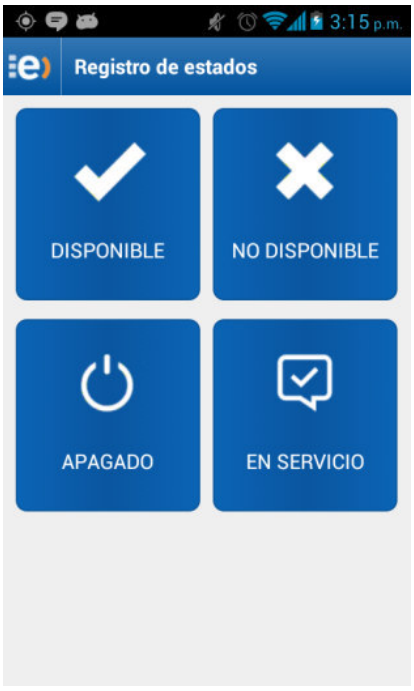
Con Págalo.pe puedes pagar las tasas de las siguientes entidades públicas:

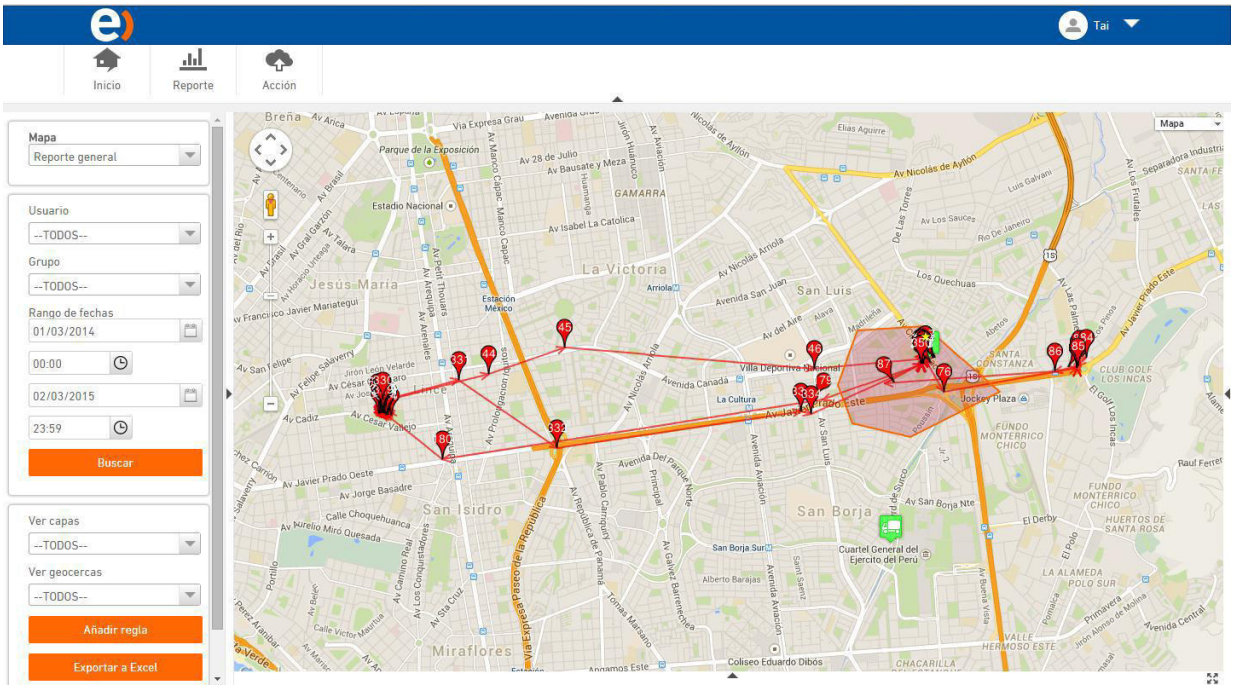
- RENIEC.
- Poder Judicial.
- Ministerio de Trabajo.
- Superintendencia Nacional de Migraciones.
- Policía Nacional del Perú.
- Ministerio de Transportes y Comunicaciones.

Seguimiento de Ruta:

Seguimiento de Ruta de Entel, es un aplicativo que permite recopilar ubicaciones geográficas de los celulares asignados a los operarios que distribuyen productos a nivel nacional, mediante la recopilación se puede mostrar en un aplicativo web usando un mapa el seguimiento en detalle de cada operario en tiempo real.






Tai
Inicio
Reporte
Acción
Mapa
Reporte general
Usuario
--TODOS--
Grupo
--TODOS--
Rango de fechas
01/03/2014
00:00
02/03/2015
23:59
Buscar
Ver capas
--TODOS--
Ver geocercas
--TODOS--
Añadir regla
Exportar a Excel
Mapa
Jesús María
La Victoria
San Luis
San Isidro
San Borja
Miraflores
Estadio Nacional
Parque de la Exposición
Estación México
Villa Deportiva Nacional
Jockey Plaza
Cuartel General del Ejercito del Perú
Coliseo Eduardo Dibós

Manual técnico de implementación de Clean Architecture en GMD.

DOCUMENTO / ENTREGABLE:

MANUAL TÉCNICO DE IMPLEMENTACIÓN DE NUEVA ARQUITECTURA ANDROID

ÁREA RESPONSABLE:

Innovación

Versión 0.2

Actualizado a 14/01/2018

HISTORIAL DE LAS REVISIONES

Item	Versión	Fecha	Autor	Descripción	Estado	Responsable de Revisión y/o Aprobación
1	0.1	10/01/2018	FS	Versión Inicial	Observado	RT
2	0.2	14/01/2018	FS	Levantamiento de observaciones	Enviado	RT

Autor(es):
FS : Albert Montes Anccasi

TABLA DE CONTENIDO

1. INTRODUCCIÓN

1.1. OBJETIVOS

El presente documento muestra la arquitectura de software en el desarrollo de aplicaciones Android, a través de diferentes vistas, cada una de las cuales ilustra un aspecto en particular del software como parte de la solución, con la finalidad de presentar una visión global y comprensible del diseño general del software desarrollado, también muestra todo lo necesario para su implementación en nuevos desarrollos.

1.2. ÁMBITO DE APLICACIÓN

El documento se centra en el desarrollo de la vista lógica del software. Se incluyen los aspectos fundamentales del resto de las vistas y se omiten aquellas que no se consideren pertinentes, como es el caso de la vista de procesos.
En cuanto a los componentes externos que se mencionen, se incluye una descripción de los mismos en el nivel que se considere apropiado y se indican las referencias donde consultar más información sobre los mismos.

2. ARQUITECTURA

2.2. OBJETIVO Y RESTRICCIONES DE ARQUITECTURA

2.2.1. Objetivos de la Arquitectura

Los objetivos de la arquitectura son los siguientes:

- Deberá mostrar independencias de capas.
- Deberá soportar desde la versión 4.0 de Android.
- Deberá tener sus componentes en módulos o librerías independientes con la finalidad de poder reutilizarlos en todos los proyectos en los que se requiera.

2.2.2. Restricciones de la Arquitectura

Se deberá tener en cuenta las siguientes restricciones:

- Infraestructura
  - ✓ Desde Android 4.0.
- Desarrollo

  Se hará uso de los siguientes frameworks y librerías para desarrollar el aplicativo:

- ✓ Utilizar el framework de inyección de vistas Butter Knife 6.0.0 en la capa de presentación (ver: http://jakewharton.github.io/butterknife/).
- ✓ Utilizar la librería de Google GSON 2.4 para serializar y deserializar Json (ver: https://github.com/google/gson).
- ✓ Utilizar la librería de Model Mapper 0.7.5 para mapear entidades y objetos (ver: http://modelmapper.org/).
- ✓ Utilizar el framework Retrofit 2.2.0 para consumir servicios web en la capa de datos (ver: http://square.github.io/retrofit/).

2.3. CAPAS ARQUITECTONICAS

En esta sección se pasa a describir cada una de las capas arquitectónicas definidas.

2.3.1. Arquitectura de Aplicaciones

En esta sección se describe detalladamente la arquitectura de la aplicación y componentes transversales.

a. Restricciones

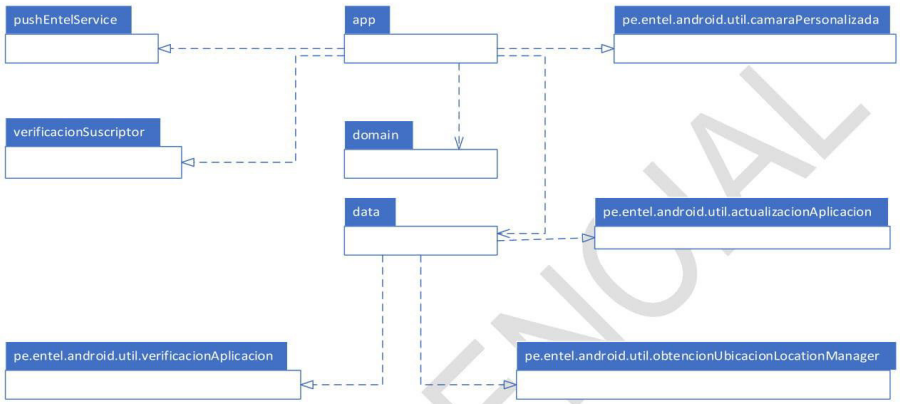
Se deberá tener en cuenta las siguientes restricciones:

- Infraestructura
  - ✓ Desde Android 4.0.
- Desarrollo

  Se hará uso de los siguientes frameworks y librerías para desarrollar el aplicativo:
  - ✓ Utilizar el framework de inyección de vistas Butter Knife 6.0.0 en la capa de presentación (ver: http://jakewharton.github.io/butterknife/).
  - ✓ Utilizar la librería de Google GSON 2.4 para serializar y deserializar Json (ver: https://github.com/google/gson).
  - ✓ Utilizar la librería de Model Mapper 0.7.5 para mapear entidades y objetos (ver: http://modelmapper.org/).
  - ✓ Utilizar el framework Retrofit 2.2.0 para consumir servicios web en la capa de datos (ver: http://square.github.io/retrofit/).
  - ✓ Utilizar el framework de acceso a datos Sugar 1.4 en la capa de datos (ver: http://satyan.github.io/sugar/).

b. Paquetes de diseño arquitectónico

Los componentes de aplicación están estructurados por un archivo APK que empaqueta los módulos. En el grafico siguiente se muestra la estructura que tendrá cada componente de aplicación.



c. Vista lógica

El diseño de la arquitectura se encuentra orientado al modelo de Arquitectura Limpia (Clean Architecture), el cual contiene las siguientes capas:

- Capa de Presentación, bajo el patrón de modelo, la vista y presentación (MVP):
  - **mapper:** Contiene el mapeo y transformación de entidades u objetos de las diferentes capas a modelos de la capa de presentación y viceversa.
  - **model:** contiene los modelos que se utilizan en esta capa.
  - **navigation:** contiene el navegador de toda la aplicación.
  - **presenter:** orquesta la interacción entre la vista y el modelo.
  - **service:** implementa los servicios utilizados en la aplicación:
  - **util:** contiene los utilitarios correspondientes a la capa.
  - **view:** contiene las vistas, estas a si mismo están organizadas en:
    - **activity:** contiene las actividades de la aplicación y donde se hace la inyección de vistas.

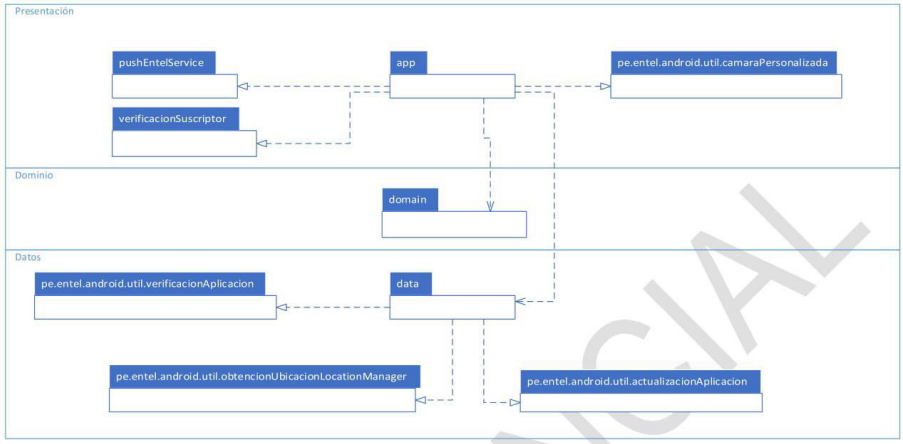
- **adapter:** contiene los adaptadores de los componentes visuales.
- **widget:** contiene widgets de la aplicación.
- **Interface:** contiene interfaces de las actividades.

- Capa de Dominio, contiene los casos de uso, entidades y las interfaces de los repositorios y está estructurado de la siguiente forma:
  - **entity:** contiene todas las entidades a utilizar en los casos de uso.
  - **exception:** contiene métodos de manejo de excepciones de la capa de dominio.
  - **repository:** contiene las interfaces de los repositorios a utilizar para cada caso de uso.
  - **usecase:** contiene los casos de uso y las interfaces del mismo.
- Capa de Datos, contiene el los componentes necesarios para el almacenado de datos locales o por medio de un servicio y está estructurado de la siguiente forma:
  - **dao:** contiene los dao que heredan de Sugar para su implementación en la base de datos.
  - **entity:** contiene las entidades de la capa de datos que no se encuentren incluidas en la implementación de Sugar.
  - **excepción:** contiene métodos de manejo de excepciones de la capa de datos.
  - **net:** contiene los métodos de conexión con los servicios, no se encuentra el consumo de cada servicio.
  - **repository:** contiene la implementación de los repositorios de dominio, el cual también se encuentra ordenado por fuente de origen (datasource).
  - **util:** contiene utilitarios de la capa de datos.

d. Vista de componentes

Está conformado por una capa de presentación representada por el **app**, una capa de dominio representada por **domain** y de acceso a datos representada por **data**, donde hay que tener en cuenta que el dominio de los componentes está compuesto por el dominio del cliente: *pe.entel.android*.

Este componente está a su vez estructurado de la siguiente forma:



2.3.2. Arquitectura de Servicios

Servicio	Punto de Decisión	Decisión
Componente de Servicio Genérico	Formato de mensaje	Json

La tecnología para el consumo de servicios web será el framework de Retrofit.

2.3.3. Arquitectura Tecnológica

No aplica.

3. IMPLEMENTACIÓN

3.1. SOLUCIÓN FINAL

Para su implementación se debe de tener 3 módulos con las siguientes características:

Nombre	Capa	Tipo
app	Presentación	Proyecto de Android
domain	Dominio	Módulo de librería java
data	Data	Módulo de librería android

Cada uno debe de tener la estructura descrita en el punto **2.2.1 c**.

3.1.1. Capa de Presentación (app)

En la capa de Presentación se debe de implementar cada uno de los siguientes paquetes, en el caso de paquetes funcionales se incluirá un ejemplo y en el caso de arquitectura se pondrá toda la implementación de los objetos:

- App.mapper **(funcional)**: contiene final class con métodos estáticos por cada Modelo que se requiera mappear o convertir. No requieren inicializar ni declarar para su uso.

  Dentro de los mapper se deben de poner los métodos para mapear entidades a Modelo, Json a Modelo o revertir en cualquier caso.

```
public final class ProductoModelMapper {


    public static ProductoModel StringAdapter(String text){
        ProductoModel model =  new ProductoModel();

        if(text != null) {
            Gson gson = new Gson();
            model = gson.fromJson(text, ProductoModel.class);
        }
        return model;
    }

    public static ProductoModel EntityAdapter(Producto entidad){
        ProductoModel model =  new ProductoModel();

        if(entidad != null) {
            ModelMapper modelMapper = new ModelMapper();
            model = modelMapper.map(entidad, ProductoModel.class);
        }
        return model;
    }

    public static Producto EntityRevert(ProductoModel model){
        Producto entidad =  new Producto();

        if(model != null) {
            ModelMapper modelMapper = new ModelMapper();
            entidad = modelMapper.map(model, Producto.class);
```

- Entidades a Modelo: en este caso se utiliza la librería de ModelMapper.

```
public static ProductoModel EntityAdapter(Producto entidad){
    ProductoModel model =  new ProductoModel();

    if(entidad != null) {
        ModelMapper modelMapper = new ModelMapper();
        model = modelMapper.map(entidad, ProductoModel.class);
    }
    return model;
}
```

- Json a Modelo: en este caso se utiliza la librería del Gson.

```
public static ProductoModel StringAdapter(String text){
    ProductoModel model =  new ProductoModel();

    if(text != null) {
        Gson gson = new Gson();
        model = gson.fromJson(text, ProductoModel.class);
    }
    return model;
}
```

- Revertir: en este caso se utiliza el ModelMapper para convertir el modelo a entidad o Gson para convertir el modelo a Json, según sea el caso.

```
public static Producto EntityRevert(ProductoModel model){
    Producto entidad =  new Producto();

    if(model != null) {
        ModelMapper modelMapper = new ModelMapper();
        entidad = modelMapper.map(model, Producto.class);
```

- App.model **(funcional)**: son clases que representa al modelo de la capa presentación:

```
public class ProductoModel {
    public final static String TAG = Producto.class.getSimpleName();;

    private Long id = null;
    private String idProducto;
    private String codigoProducto;
    private String nombre;
    private String precio;
    private String cantidad;
    private String monto;

    public String getIdProducto() { return idProducto; }

    public void setIdProducto(String idProducto) { this.idProducto = idProducto; }

    public String getCodigoProducto() { return codigoProducto; }

    public void setCodigoProducto(String codigoProducto) { this.codigoProducto = codigoProducto; }

    public String getNombre() { return nombre; }
```

```
public class MenuModel {

    public final static int SINCRONIZAR = 0;
    public final static int SALIR = 1;
    public final static int INICIO = 2;
    public final static int PENDIENTE = 3;
    public final static int FOTO = 4;

    int id;
    String titulo;
    int imagen;

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getTitulo() { return titulo; }

    public void setTitulo(String titulo) { this.titulo = titulo; }

    public int getImagen() { return imagen; }
```

- App.navigation **(funcional)**: es una final class, con métodos estáticos que no requiere su inicialización ni implementación, sus métodos deben de recibir el contexto y parámetros a enviar en la navegación:

```
public final class Navigator {

    public static void navigateToLogin(Context context){
        if(context != null){
            Intent intent = LoginActivity.getCallingIntent(context);
            context.startActivity(intent);
        }
    }

    public static void navigateToFoto(Context context){
        if(context != null){
            Intent intent = FotoActivity.getCallingIntent(context);
            ((Activity)context).startActivityForResult(intent, EntelConfig.CONSIDCAMARA);
        }
    }

    public static void navigateToMensaje(Context context){
        if(context != null){
            Intent intent = MensajeActivity.getCallingIntent(context);
            context.startActivity(intent);
        }
    }

    public static void navigateToMenu(Context context){
        if(context != null){
            Intent intent = MenuActivity.getCallingIntent(context);
            ((Activity)context).finish();
            context.startActivity(intent);
```

- App.presenter **(funcional)**: son clases que contienen la interacción de la vista con el modelo, también es el que se comunica con la capa de dominio y presentación:

```
public class LoginPresenter {
    final LoginView view;
    public LoginPresenter(LoginView view) { this.view = view; }

    public void iniciar() {
        obtenerVersion();
        verificarUsuario();
    }

    private void verificarUsuario() {
        UsuarioRepository usuarioRepository = new UsuarioDataRepository(view.getContext());
        GetUsuarioUseCase getUsuarioUseCase = new GetUsuarioUseCaseImpl(usuarioRepository);
        UsuarioModel usuario = UsuarioModelMapper.adapter(getUsuarioUseCase.ejecutar());

        if(usuario!=null) {
            view.actualizarUsuario(usuario.getCodigo(), usuario.getClave());
        }
    }

    private void obtenerVersion() {
```

- App.service **(funcional)**: son las clases que contienen los servicios a utilizar en cada aplicación, como las notificaciones y push:

```
public class MensajeNotifier extends IntentService {

    private NotificationManager ioNotifyManager;
    public final static int CODIGO_NOTIFICACION_ESTADO = 133707736;

    public MensajeNotifier() { super("MensajeNotifier"); }

    @Override
    public void onCreate() {
        super.onCreate();
        ioNotifyManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    }

    @Override
    protected void onHandleIntent(Intent intent) {

        String lsMensaje = intent.getStringExtra(EntelConfig.PUSH_BUNDLE_MENSAJE).substring(1);
        String lsCodMensaje = intent.getStringExtra(EntelConfig.PUSH_BUNDLE_CODMENSAJE);
        showNotificacion(lsMensaje, true);
    }

    public void showNotificacion(String mensaje, boolean flgSonido) {
        Intent loIntent = new Intent(this, MensajeActivity.class);
        loIntent.putExtra(EntelConfig.PUSH_BUNDLE_MENSAJE,mensaje);

        int liCodigoNotificacion = CODIGO_NOTIFICACION_ESTADO;
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, loIntent,
                //Intent.FLAG_ACTIVITY_REORDER_TO_FRONT
                // |
                PendingIntent.FLAG_UPDATE_CURRENT
                        | PendingIntent.FLAG_ONE_SHOT
        );
```

```
public class PushBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context poContext, Intent poIntent) {
        Bundle loBundle = poIntent.getExtras();

        String topic = loBundle.getString(EntelConfig.PUSH_BUNDLE_TOPICNAME);
        String topicValidacion = UtilitarioApp.fnReadParamProperty(poContext, "MQTT_TOPIC

        Log.v("PUSH", "topic: " + topic);
        Log.v("PUSH", "topicValidacion: " + topicValidacion);

        if (topicValidacion.equals(topic)) {
            String codigo = loBundle.getString(EntelConfig.PUSH_BUNDLE_CODMENSAJE);
            String mensaje = loBundle.getString(EntelConfig.PUSH_BUNDLE_MENSAJE);
            Log.v("PUSH", "codigo: " + codigo);
            Log.v("PUSH", "mensaje: " + mensaje);

            Intent loIntent = new Intent(poContext, MensajeNotifier.class);
            loIntent.putExtra(EntelConfig.PUSH_BUNDLE_MENSAJE,mensaje);
            loIntent.putExtra(EntelConfig.PUSH_BUNDLE_CODMENSAJE,codigo);
            poContext.startService(loIntent);

        }

    }
}
```

- App.util **(funcional)**: se encuentran los utilitarios de la aplicación, siendo recomendables los siguientes:
  - CustomComparator

```
public class CustomComparator implements Comparator<Size> {

    @Override
    public int compare(Size o1, Size o2) {
        if (o1.width < o2.width) return 1;
        else return -1;

    }
}
```

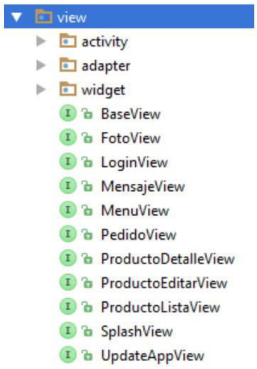
  - EntelConfig

```
public class EntelConfig {

    public static String CONSPROPERTYFILE = "params.properties";
    public static final int CONSIDCAMARA = 1337;
    public static final String IDENTIFICADOR = "IDENTIFICADOR";
    public final static String PUSH_BUNDLE_TOPICNAME = "bunTopicName";
    public final static String PUSH_BUNDLE_MENSAJE = "bunMensaje";
    public final static String PUSH_BUNDLE_CODMENSAJE = "bunCodMensaje";


    public static enum EnumRolPreferencia {
        /**
         * Rol de Administrador. Puede modificar las preferencias
         */
        ADMINISTRADOR,
        /**
         * Rol de Usuario. Puede visualizar las preferencias
         */
        USUARIO
    };
}
```

- UtilitarioApp

```
public class UtilitarioApp {

    private static final String TAG = "UtilitarioApp";

    public static String CONSPROPERTYFILE = "params.properties";

    public static String fnVersion(Context poContext) {
        String lsVersion = "";
        PackageInfo loPackageInfo = null;
        try {
            loPackageInfo = poContext.getPackageManager().getPackageInfo(
                    poContext.getPackageName(), 0);
        } catch (PackageManager.NameNotFoundException e) {
            e.printStackTrace();
            Log.v(TAG, "Verificar el manifest: " + e.getMessage());
            return "ERROR VERSION";
        }

        lsVersion = loPackageInfo.versionName;

        return lsVersion;
    }

    public static Properties readProperties(Context ctx) {
        return readProperties(ctx, EntelConfig.CONSPROPERTYFILE);
    }

    public static Properties readProperties(Context ctx, String arc) {
        Properties prop = null;
        try {
            AssetManager am = ctx.getAssets();
            InputStream is = am.open(arc);
            prop = new Properties();
            prop.load(is);
        } catch (Exception ex) {
            Log.e(TAG, "readProperties: ", ex);
        }
        return prop;
    }
```

- App.view **(funcional)**: se encuentran en esta raíz todas las interfaces de las actividades:



```
public interface FotoView extends BaseView{

    void mostrarPreview();

    void hideCamera();

    void preview(Bitmap bMapRotate);

    void saveFotoOk();

    void mostrarEnvio();

    void saveFotoError();

    void showCamera();
}
```

- App.view.activity **(funcional)**: se encuentran todas las actividades que implementan las interfaces descritas en el punto anterior, también se realiza la inyección de vistas:

```
public class FotoActivity extends BaseActivity implements FotoView , CameraFragment.PerformerCallback, CameraFragment.TipoCamaraCa

    FotoPresenter presenter;

    @InjectView(R.id.toma_foto_ib_enviar)
    ImageButton ibEnviarFoto;
    @InjectView(R.id.toma_foto_iv_preview)
    ImageView ivPreview;

    CameraFragment cameraFragment;

    private boolean grabada = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_foto);
        ButterKnife.inject(this);
        cameraFragment = (CameraFragment)getSupportFragmentManager().findFragmentById(R.id.toma_foto_camera_fragment);
        presenter = new FotoPresenter(this);
    }

    @Override
```

- App.view.adapter **(funcional)**: se encuentran todos los adaptadores de los componentes visuales.

```
public class MenuPrincipalAdapter extends ArrayAdapter {

    int resource;


    public MenuPrincipalAdapter(Context context, int resource, List<MenuModel> list) {
        super(context, resource, list);
        this.resource = resource;

    }

    public View getView(int position, View convertView, ViewGroup parent) {
        LinearLayout nuevaVista;
        LayoutInflater inflater = (LayoutInflater) getContext()
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        if (convertView == null) {
            nuevaVista = new LinearLayout(getContext());
            inflater.inflate(resource, nuevaVista, true);

        } else {
            nuevaVista = (LinearLayout) convertView;
        }
```

- App.view.widget **(funcional)**: se encuentran todos los widget de la aplicación.

```
public class AppWidget extends AppWidgetProvider {

    static void updateAppWidget(Context context, AppWidgetManager appWidgetManager,
                                int appWidgetId) {

        CharSequence widgetText = "EXAMPLE";
        RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.app_widget);
        views.setTextViewText(R.id.appwidget_text, widgetText);

        appWidgetManager.updateAppWidget(appWidgetId, views);
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        for (int appWidgetId : appWidgetIds) {
            updateAppWidget(context, appWidgetManager, appWidgetId);
        }
    }
```

3.1.2. Capa de Dominio (domian)

En la capa de Dominio se debe de implementar cada uno de los siguientes paquetes, en el caso de paquetes funcionales se incluirá un ejemplo y en el caso de arquitectura se pondrá toda la implementación de los objetos:

- Domain.entity **(funcional)**: se encuentran todas la entidades del dominio:

```
public class Localizacion {

    private String latitud;
    private String longitud;
    private String gpsPrecision;
    private String velocidad;
    private String fechaMovil;
    private String gpsOrigen;


    public String getLatitud() { return latitud; }

    public void setLatitud(String latitud) { this.latitud = latitud; }

    public String getLongitud() { return longitud; }

    public void setLongitud(String longitud) { this.longitud = longitud; }

    public String getVelocidad() { return velocidad; }

    public void setVelocidad(String velocidad) { this.velocidad = velocidad; }
```

- Domain.exception **(arquitectura)**: contiene los métodos de manejo de excepciones de la capa de Dominio:
  - EncapsulatedErrorBundle

```
public class EncapsulatedErrorBundle implements ErrorBundle {

    private Exception error;

    public EncapsulatedErrorBundle(Exception e) { this.error = e; }

    @Override
    public Exception getException() { return error; }

    @Override
    public String getErrorMessage() {
        if (error != null)
            return error.getMessage();
        else
            return "No definido";
    }
}
```

- ErrorBundle

```
public interface ErrorBundle {
    Exception getException();

    String getErrorMessage();
}
```

- NotExistErrorBundle

```
public class NotExistErrorBundle implements ErrorBundle {

    private String nombre;

    public NotExistErrorBundle(String nombre) { this.nombre = nombre; }

    @Override
    public Exception getException() { return new Exception("Not Exist " + nombre); }

    @Override
    public String getErrorMessage() { return "Not exist " + nombre; }
}
```

- Domain.repository **(funcional)**: contiene las interfaces de los repositorios a implementar en la capa de datos:

```
public interface FotoRepository {

    void guardarFoto(String idUsuario, String identificador, byte[] data, GuardarFotoCallback guardarFotoCallback);

    void enviarFoto(String identificador, EnviarFotoCallback enviarFotoCallback);

    void eliminarFoto(String identificador);

    String obtenerRutaFoto(String identificador);

    int getTipoFoto();

    void setTipoFoto(int tipo);

    List<Photo> listarFotosPendientes();

    interface GuardarFotoCallback {
        void onGuardado();
        void onError(ErrorBundle errorBundle);
    }

    interface EnviarFotoCallback{
```

- Domain.usecase **(funcional)**: contiene todas las interfaces de los casos de uso:

```
public interface ListaPedidoPendientesUseCase {
    List<Pedido> ejecutar();
}
```

- Domain.usecase.implementacion **(funcional)**: contiene la implementación de los caso de uso:

```
public class EnviarPedidoUseCaseImpl implements EnviarPedidoUseCase {

    final PedidoRepository pedidoRepository;
    final LocationRepository locationRepository;
    final UsuarioRepository usuarioRepository;

    public EnviarPedidoUseCaseImpl(PedidoRepository pedidoRepository, LocationRepository locationRepository, UsuarioRepository u
        this.pedidoRepository = pedidoRepository;
        this.locationRepository = locationRepository;
        this.usuarioRepository = usuarioRepository;
    }

    @Override
    public void ejecutar(final Pedido pedido, final Callback callback) {

        locationRepository.obtenerLocalizacion((localizacion) -> {
                enviarPedido(pedido,localizacion,callback);
        });

    }
```

3.1.3. Capa de Data (data)

En la capa de Data se debe de implementar cada uno de los siguientes paquetes, en el caso de paquetes funcionales se incluirá un ejemplo y en el caso de arquitectura se pondrá toda la implementación de los objetos:

- Data.dao **(funcional)**: contiene las entidades a utilizar en la implementación del Sugar:

```
public class FotoDao extends SugarRecord {

    private String idUsuario;
    private String identificador;
    private String flgEnviado;

    public String getIdentificador() { return identificador; }

    public void setIdentificador(String identificador) { this.identificador = identificador; }

    public String getIdUsuario() { return idUsuario; }

    public void setIdUsuario(String idUsuario) { this.idUsuario = idUsuario; }

    public String getFlgEnviado() { return flgEnviado; }

    public void setFlgEnviado(String flgEnviado) { this.flgEnviado = flgEnviado; }
}
```

- Data.entity **(funcional)**: contiene las entidades que no se utilizan en la implementación del Sugar:

```
public class FotoEntity extends PhotoEntity{

    private static final String TAG = FotoEntity.class.getName();

    private String idUsuario;
    private byte[] imagen;

    public String getIdUsuario() { return idUsuario; }

    public void setIdUsuario(String idUsuario) { this.idUsuario = idUsuario; }

    @Override
    public void agregarDatos(DataOutputStream outputStream) throws IOException {
        Log.v(TAG,"identificador"+this.identificador);
        Log.v(TAG,"idUsuario"+this.idUsuario);
        outputStream.writeLong(Long.parseLong(this.identificador));
        outputStream.writeInt(Integer.parseInt(this.idUsuario));
    }


}
```

- Data.exception **(arquitectura)**: contiene los métodos de manejo de excepciones de la capa de Datos:
  - NetworkConnectionException

```
public class NetworkConnectionException extends Exception {

  public NetworkConnectionException() { super(); }

  public NetworkConnectionException(final String message) { super(message); }

  public NetworkConnectionException(final String message, final Throwable cause) {
    super(message, cause);
  }

  public NetworkConnectionException(final Throwable cause) { super(cause); }
}
```

  - RepositoryErrorBundle

```
public class RepositoryErrorBundle implements ErrorBundle {

    private final Exception exception;

    public RepositoryErrorBundle(Exception exception) { this.exception = exception; }

    public Exception getException() { return exception; }

    public String getErrorMessage() {
        String message = "";
        if (this.exception != null) {
            message = this.exception.getMessage();
        }
        return message;
    }
}
```

  - UserNotFoundException

```
public class UserNotFoundException extends Exception {

    public UserNotFoundException() { super(); }

    public UserNotFoundException(final String message) { super(message); }

    public UserNotFoundException(final String message, final Throwable cause) {
        super(message, cause);
    }

    public UserNotFoundException(final Throwable cause) { super(cause); }
}
```

- Data.repository **(funcional)**: contiene la implementación de los repositorios, descritos en la capa de dominio:

```
public class FotoDataRepository implements FotoRepository {

    final Context context;

    public FotoDataRepository(Context context) { this.context = context; }

    @Override
    public void guardarFoto(final String idUsuario, final String identificador, byte[] data, final GuardarFotoCallback guardar
        FileStore fileStore = new FileStore(context);
        FotoApplicationStore applicationStore = new FotoApplicationStore(context, fileStore);
        final FotoLocalStore fotoLocalStore = new FotoLocalStore();

        applicationStore.guardarFoto(identificador, data, new FotoApplicationStore.GuardarFotoCallback() {
            @Override
            public void onGuardado() {
                fotoLocalStore.guardarFoto(idUsuario,identificador);
                guardarFotoCallback.onGuardado();
            }

            @Override
            public void onError(Exception e) {
                guardarFotoCallback.onError(new RepositoryErrorBundle(e));
            }
        });
    }
```

- Data.repository.datasource.application **(funcional)**: contiene los métodos para registrar en el dispositivo móvil.

```
public class FotoApplicationStore {

    final Context context;
    final FileStore fileStore;
    final String raiz;

    public FotoApplicationStore(Context context, FileStore fileStore) {
        this.context = context;
        this.fileStore = fileStore;
        raiz = UtilitarioData.getRaiz(context);
    }

    public void guardarFoto(String identificador, byte[] data, GuardarFotoCallback guardarFotoCallback ){

        try {
            fileStore.storeFileImage(data,raiz,identificador);
        } catch (Exception e) {
            guardarFotoCallback.onError(e);
        }

        guardarFotoCallback.onGuardado();
    }

    public String obtenerFotoRuta(String identificador){
        return fileStore.fnRutGrabado(raiz,identificador);
    }
```

- Data.repository.datasource.cloud **(funcional)**: contiene los métodos para conectarse con los servicios web.

```
public class FotoCloudStore {

    final Context context;
    EnviarFotoCallback enviarFotoCallback;


    public FotoCloudStore(Context context) { this.context = context; }


    public void enviarFoto(FotoEntity fotoEntity, final EnviarFotoCallback enviarFotoCallback ){
        this.enviarFotoCallback = enviarFotoCallback;

        if (UtilitarioData.isThereInternetConnection(context)) {
            Gson gson = new Gson();

            Intent intent = new Intent(context, BinaryService.class);
            intent.setData(Uri.parse(Url.obtenerRuta(context,
                    Url.EnumUrl.FOTO)));
            Log.v("URL", Url.obtenerRuta(context, Url.EnumUrl.FOTO));
            intent.putExtra(BinaryService.EXTRA_ONLY_A_PARAM, gson.toJson(fotoEntity));
            intent.putExtra(BinaryService.EXTRA_RESULT_RECEIVER, enviarFoto.getResultReceiver());
            context.startService(intent);
        } else {
            enviarFotoCallback.onError(new NetworkConnectionException("Fuera de cobertura"));
        }

    }
```

- Data.repository.datasource.local **(funcional)**: contiene los métodos para registrar en la base de datos de la aplicación.

```
public class FotoLocalStore {

    public void guardarFoto(String idUsuario, String identificador){
        FotoDao fotoDao = new FotoDao();
        fotoDao.setIdUsuario(idUsuario);
        fotoDao.setIdentificador(identificador);
        fotoDao.setFlgEnviado("F");
        fotoDao.save();
    }

    public void borrarFoto(String identificador){
        FotoDao.deleteAll(FotoDao.class,"identificador = ?",identificador);
    }

    public FotoDao obtenerFoto(String identificador){
        return FotoDao.find(FotoDao.class,"identificador = ?",identificador).get(0);
    }

    public void guardarFoto(FotoDao fotoDao) { fotoDao.save(); }

    public List<FotoDao> listarFotosPendientes(){
        return FotoDao.find(FotoDao.class,"flg_enviado = 'F'");
    }

}
```

- Data.repository.datasource.preference **(funcional)**: contiene los métodos para registrar en las preferencias de la aplicación.

```
public class FotoPreferenceStore extends PreferenceStore {

    private final static String TIPO_CAMARA ="TIPO_CAMARA";

    public FotoPreferenceStore(Context context) { super(context); }

    public void setTipoCamara(int tipo) {
        saveOnSharePreferences(TIPO_CAMARA,tipo);

    }

    public int getTipoCamara() { return getPreferenceInt(TIPO_CAMARA); }
}
```

3.1.4. Conexión de capas

En el paquete **presenter** de la capa de **Presentación**, se definen cada una de las interacciones de la vista, modelo, dominio y datos.

En el caso de conectar la capa de **Presentación** a la capa de **Datos** para enviar a su respectivo caso de uso se debe de hacer lo siguiente:

```
UsuarioRepository usuarioRepository = new UsuarioDataRepository(view.getContext());
```

Donde UsuarioRepository representa la interface del repositorio descrita en el paquete **domain.repository**, siendo esta implementada por UsuarioDataRepositorio, el cual se encuentra en el paquete **data.repository**, la implementación suele recibir el contexto de la aplicación para su uso en la capa de datos.

En el caso de conectar la capa de **Presentación** con la capa de **Dominio** para ejecutar sus casos de uso se debe de hacer lo siguiente:

```
GuardarFotoUseCase guardarFotoUseCase = new GuardarFotoUseCaseImpl(fotoRepository, usuarioRepository);
```

Donde GuardarFotoUseCase representa la interface del caso de uso descrita en el paquete **domain.usecase**, siendo esta implementada por GuardarFotoUseCaseImpl, el cual se encuentra en el paquete **domain.usecase.implementation**, la implementación suele recibir las interfaces de los repositorios previamente instanciados.

3.1.5. Uso del Sugar

En el proyecto de la capa **data** se debe de incluir la dependencia del **Sugar** dentro del build.gradle:

```
compile 'com.github.satyan:sugar:1.4'
```

Tambien se debe de incluir el atrubuto **name** en el **AndroidManifest.xml** de la capa de datos, con el valor **com.orm.SugarApp**, para iniciar la funcionalidad del Sugar en la aplicación.

Creación de tablas-entidades

Se debe de crear entidades las cuales extiendan del **SugarRecord**:

```
public class FotoDao extends SugarRecord {

    private String idUsuario;
    private String identificador;
    private String flgEnviado;

    public String getIdentificador() { return identificador; }

    public void setIdentificador(String identificador) { this.identificador = identificador; }

    public String getIdUsuario() { return idUsuario; }

    public void setIdUsuario(String idUsuario) { this.idUsuario = idUsuario; }

    public String getFlgEnviado() { return flgEnviado; }

    public void setFlgEnviado(String flgEnviado) { this.flgEnviado = flgEnviado; }
}
```

Métodos principales

Para las búsquedas en las tablas se debe de hacer uso del método **find** del **Sugar**, con sus respectivos filtros:

```
FotoDao.find(FotoDao.class,"flg_enviado = 'F'");
```

Para la eliminación de registros en las tablas se debe de hacer uso del método **deleteAll** del **Sugar**, con sus respectivos filtros:

```
FotoDao.deleteAll(FotoDao.class,"identificador = ?",identificador);
```

Para las grabar en las tablas se debe de hacer uso del método **save** del **Sugar**, después de haber seteado los atributos de la entidad:

```
FotoDao fotoDao = new FotoDao();
fotoDao.setIdUsuario(idUsuario);
fotoDao.setIdentificador(identificador);
fotoDao.setFlgEnviado("F");
fotoDao.save();
```

Para mayor información ver: http://satyan.github.io/sugar/getting-started.html.

3.1.6. Uso del Retrofit

En el proyecto de la capa **data** se debe de incluir la dependencia del **Retrofit** dentro del build.gradle:

```
compile 'com.squareup.retrofit2:retrofit:2.2.0'
compile 'com.squareup.retrofit2:converter-gson:2.2.0'
```

Se debe de iniciar el **Retrofit** en los datasource de **data.repository.datasource.cloud**:

```
public class PedidoCloudStore {
    final Context context;
    final Retrofit retrofit;

    public PedidoCloudStore(Context context) {
        this.context = context;
        retrofit = new Retrofit.Builder()
                .baseUrl(Url.obtenerBase(context))
                .addConverterFactory(GsonConverterFactory.create())
                .build();
    }
```

Para mayor información de cada uno de los métodos: http://square.github.io/retrofit.

3.2. SOLUCIÓN SUGERIDA:

El equipo de GMD sugirió en primera instancia el uso del patrón IoC para potenciar la arquitectura Clean y la inyección de vistas que actualmente tiene, dicho patrón cumple con el principio 5 del esquema SOLID, el de inversión de dependencias, siendo la inyección de dependencias solo un método de dicho principio (ver: https://en.wikipedia.org/wiki/Dependency_inversion_principle).

La herramienta propuesta por nuestra parte y aprobada en primera instancia por Entel es el Dagger de Android (ver: http://square.github.io/dagger/), el cual tiene las siguientes características aparte de las inyecciones de dependencias:

- Uso de anotaciones: usa etiquetas para la inyección de dependencias.
- Creación de módulos y scope: permiten poner métodos y objetos en diferentes niveles de uso, los cuales al llevar algunos métodos de las actividades o presentadores hace que estos sean más eficientes, dado que su tiempo de uso en la aplicación es definida por el programador y no solo por la memoria del dispositivo.

Nosotros en GMD usamos y recomendamos este patrón y herramienta para el desarrollo de aplicaciones móviles, los cuales solo nos han traído relativo impacto en las capacitaciones, pero no en el desarrollo del software, haciendo que el control de aplicativos, así como el rendimiento de los mismos sean ajustados y fácilmente optimizados, permitiéndonos crear aplicaciones con grandes flujos de navegación y datos, evitando los retrabajos y reformulamientos.

Presentación de propuesta al cliente

Índice

Nuestra **Agenda**

- Introducción
- Alcance
- Cambios
- Esquema Final

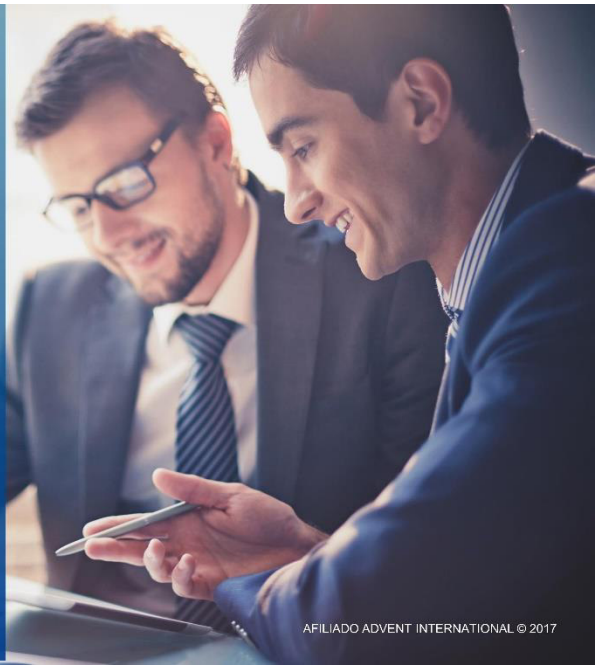
GMD®

Como se encuentra actualmente?

Actualmente la arquitectura Android de Entel, contempla 3 proyectos:

- **App (capa de presentación, patrón MVP).**
- **Domain (capa de dominio, casos de uso).**
- **Data (capa de datos, conexión a repositorios internos o externos).**

GMD



ALCANCE

- Actualizar la arquitectura de Entel con las recomendaciones presentadas en el documento *GMD - Entel - Documento de Recomendaciones de Arquitectura Android Entel v0 1.docx*, con fecha 06 de Julio del 2017.
- Adicionando las revisiones de la cámara desarrollada por Entel y su método de Localización.

Recomendaciones &

Implementación

"Se recomienda tener sus propias entidades y hacer la relación de ambas entidades de diferentes capas por medio de adaptadores (Capa de Presentación)".

- Dentro del paquete **pe.entel.android.plantilla.model**, se contemplaron las entidades que se expondrán en la capa de presentación.
- Las clases que adaptan las entidades de otras capas con estos modelos se encuentran en el nuevo paquete **pe.entel.android.plantilla.mapper.**

Recomendaciones &

Implementación

"Se recomienda separar las interfaces y clases que lo implementan en carpetas o paquetes diferentes, ello para mantener un mayor orden y relación entre ambos (Capa de Dominio)".

- Dentro del paquete **pe.entel.android.plantilla.domain.usecase** se creo el paquete **implementation**.
- Dentro del paquete **pe.entel.android.plantilla.domain.usecase .implementation**, se encuentra la implementación de cada uno de los casos de uso y dentro del paquete **pe.entel.android.plantilla.domain.usecase** se encuentra la interface de cada caso de uso.

Recomendaciones &

Implementación

"Agrupar en paquetes las fuentes de dato, como por ejemplo todas clases y métodos que se almacenan en la nube en un paquete, las que almacenan en preferencias en otro, para evitar confusiones y ayuda a reutilizar (Capa de Datos)".

- Dentro del paquete **pe.entel.android.plantilla.data.repository.datasource** se crearon los paquetes de **application, cloud, local, preference**.
- Dentro del paquete **pe.entel.android.plantilla.data.repository.datasource.application**, se encuentran todas las clases que obtienen o envían su información del mismo dispositivo.
- Dentro del paquete **pe.entel.android.plantilla.data.repository.datasource.cloud**, se encuentran todas las clases que obtienen o envían su información por medio de la red, como servicios web.
- Dentro del paquete **pe.entel.android.plantilla.data.repository.datasource.local**, se encuentran todas las clases que obtienen o envían su información de la base de datos de la aplicación.
- Dentro del paquete **pe.entel.android.plantilla.data.repository.datasource.preference**, se encuentran todas las clases que obtienen o envían su información de las preferencias de la aplicación.

Recomendaciones &

Implementación

"Se recomienda que se use el patrón IoC (Inversión de Control)".

- Se implemento el patrón IoC dentro de la capa de presentación.
- El framework empleado fue el Dagger 2.
- Se creo el paquete **pe.entel.android.plantilla.dagger**, el cual contiene objetos y métodos de configuración y uso.
- Se creo el paquete **pe.entel.android.plantilla.dagger.component**, donde se encuentra el componente **SystemComponent** el cual esta relacionado al modulo **SystemModule** y se tiene un scope de nivel **Singleton**, el cual solo maneja una instancia.
- Se creo la clase **AndroidApplication**, la cual extiende de **SugarApp** y esta de **Application**, lo cual nos permite iniciar el Sugar requerido para la capa de datos e iniciar el Dagger.

GMD®
AFILIADO ADVENT INTERNATIONAL © 2017

Recomendaciones &

Implementación

"Se recomienda que se use el patrón IoC (Inversión de Control)".

- Se creo el paquete **pe.entel.android.plantilla.dagger.module**, donde se encuentran los módulos de **SystemModule** (modulo principal) y **UsuarioModule** (modulo de ejemplo).
- Se creo el paquete **pe.entel.android.plantilla.dagger.scope**, donde se encuentran los scopes personalizados, como ejemplo se encuentra el de **UsuarioScope**.
- Se creo el paquete **pe.entel.android.plantilla.dagger.subcomponent**, donde se encuentran los subcomponentes creados, como ejemplo se encuentra el de **UsuarioComponent.**
- Se centralizó las navegaciones en un solo objeto, para ello se creo el paquete **pe.entel.android.plantilla.navigation**, donde se encuentra la clase **Navegator**, el cual contiene todos los métodos de navegación de la aplicación, esta clase tiene el scope de Singleton, el cual solo maneja una sola instancia.

Recomendaciones &

Implementación

"Necesitamos que nos propongan, en el ejemplo que se va implementar, una mejor forma de tomar de fotos; considerando las restricciones que tiene el ejemplo" - Tamayo Valdivia, Robert.

- Funcionalidad encontrada:
  - Permite alternar entre cámaras (delantera y trasera).
  - Captura la imagen en arreglo de bytes.
- Restricciones encontradas:
  - No permite hacer zoom de las fotos.
  - No permite activar flash en las fotos.
  - No permite enfocar manualmente las fotos.
  - Tienen un tamaño máximo.
- Conclusiones:
  - Se recomienda utilizar la cámara nativa del dispositivo en la mayoría de casos, salvo en casos de incluir restricciones, como se detecto.
  - En la actual plantilla la cámara cumple con los requisitos de una cámara personalizada y esos son el estar implementado con fragmentos, capturar la imagen en un arreglo de bytes, tener los métodos de control implementados.
  - los cambios a incluir en caso no se requiera agregar funcionalidad solo son de diseño.

GMD®
AFILIADO ADVENT INTERNATIONAL © 2017

Recomendaciones &

Implementación

"Necesitamos que nos propongan una mejor forma de obtener la posición GPS desde el equipo móvil." - Tamayo Valdivia, Robert.

- Para obtener la ubicación del dispositivo se puede hacer por medio de la clase **LocationManager**, la misma que se encuentran usando.
- Si usan algún servicio de mapas pueden obtener la ubicación por medio del servicio, como por ejemplo Google Maps lo cual es recomendable en dichos casos dado que la actualización de la ubicación con el punto dibujado en el mapa es casi inmediata, en este caso quedamos a la espera de la conformidad de si desean incluir en la plantilla un ejemplo con mapas como Google Maps.
- De acuerdo del uso del **LocationManager**, solo se encontró redundancia de código en algunos casos, los cuales serán expuestos al final de la presentación. Esto no influye en la funcionalidad.

Recomendaciones &

Implementación

"Dejamos a su criterio que funcionalidades se pueden extraer de la aplicación y como las agruparían en diferentes librerías." - Tamayo Valdivia, Robert.

- Se consideraron las siguientes librerías:
  - PresenterUtil.jar, incluye los utilitarios de la capa de presentación:
    - *CameraSurface*
    - *CustomComparator*
    - *UtilitarioApp*
  - DataUtil.jar, incluye los utilitarios de la capa de datos:
    - *FotoDataRepository*
    - *LocationDataRepository*
    - *OperadorDataRepository*
    - *UtilDataRepository*
    - *TrafficMonitor*

GMD®
AFILIADO ADVENT INTERNATIONAL © 2017